Statistical machine learning for information retrieval

Adam Berger

April, 2001

CMU-CS-01-110

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
John Lafferty, Chair
Jamie Callan
Jaime Carbonell
Jan Pedersen (Centrata Corp.)
Daniel Sleator

This research was supported in part by NSF grants IIS-9873009 and IRI-9314969, DARPA AASERT award DAAH04-95-1-0475, an IBM Cooperative Fellowship, an IBM University Partnership Award, a grant from JustSystem Corporation, and by Claritech Corporation.

Keywords
Information retrieval, machine learning, language models, statistical inference, Hidden Markov Models, information theory, text summarization

Dedication

I am indebted to a number of people and institutions for their support while I conducted the work reported in this thesis.

IBM sponsored my research for three years with a University Partnership and a Cooperative Fellowship. I am in IBM's debt in another way, having previously worked for a number of years in the automatic language translation and speech recognition departments at the Thomas J. Watson Research Center, where I collaborated with a group of scientists whose combination of intellectual rigor and scientific curiosity I expect never to find again. I am also grateful to Claritech Corporation for hosting me for several months in 1999, and for allowing me to witness and contribute to the development of real-world, practical information retrieval systems.

My advisor, colleague, and sponsor in this endeavor has been John Lafferty. Despite our very different personalities, our relationship has been productive and (I believe) mutually beneficial. It has been my great fortune to learn from and work with John these past years.

This thesis is dedicated to my family: Rachel, for her love and patience, and Jonah, for finding new ways to amaze and amuse his dad every day.

Abstract

The purpose of this work is to introduce and experimentally validate a framework, based on statistical machine learning, for handling a broad range of problems in information retrieval (IR).

Probably the most important single component of this framework is a parametric statistical model of word relatedness. A longstanding problem in IR has been to develop a mathematically principled model for document processing which acknowledges that one sequence of words may be closely related to another even if the pair have few (or no) words in common. The fact that a document contains the word `automobile`, for example, suggests that it may be relevant to the queries `Where can I find information on motor vehicles?` and `Tell me about car transmissions`, even though the word `automobile` itself appears nowhere in these queries. Also, a document containing the words `plumbing, caulk, paint, gutters` might best be summarized as `common house repairs`, even if none of the three words in this candidate summary ever appeared in the document.

Until now, the word-relatedness problem has typically been addressed with techniques like automatic query expansion [75], an often successful though *ad hoc* technique which artificially injects new, related words into a document for the purpose of ensuring that related documents have some lexical overlap.

In the past few years have emerged a number of novel probabilistic approaches to information processing—including the language modeling approach to document ranking suggested first by Ponte and Croft [67], the non-extractive summarization work of Mittal and Witbrock [87], and the Hidden Markov Model-based ranking of Miller *et al.* [61]. This thesis advances that body of work by proposing a principled, general probabilistic framework which naturally accounts for word-relatedness issues, using techniques from statistical machine learning such as the Expectation-Maximization (EM) algorithm [24]. Applying this new framework to the problem of ranking documents by relevancy to a query, for instance, we discover a model that contains a version of the Ponte and Miller models as a special case, but surpasses these in its ability to recognize the relevance of a document to a query even when the two have minimal lexical overlap.

Historically, information retrieval has been a field of inquiry driven largely by empirical considerations. After all, whether system A was constructed from a more sound theoretical framework than system B is of no concern to the system's end users. This thesis honors the strong engineering flavor of the field by evaluating the proposed algorithms in many different settings and on datasets from many different domains. The result of this analysis is an empirical validation of the notion that one can devise useful real-world information processing systems built from statistical machine learning techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The purpose of this document is to substantiate the following assertion: statistical machine learning represents a principled, viable framework upon which to build high-performance information processing systems. To prove this claim, the following chapters describe the theoretical underpinnings, system architecture and empirical performance of prototype systems that handle three core problems in information retrieval.

The first problem, taken up in Chapter 3, is to assess the relevance of a document to a query. "Relevancy ranking" is a problem of growing import: the remarkable recent increase in electronically available information makes finding the most relevant document within a sea of candidate documents more and more difficult, for people and for computers. This chapter describes an automatic method for learning to separate the wheat (relevant documents) from the chaff. This chapter also contains an architectural and behavioral description of WEAVER, a proof-of-concept document ranking system built using these automatic learning methods. Results of a suite of experiments on various datasets—news articles, email correspondences, and user transactions with a popular web search engine—suggest the viability of statistical machine learning for relevancy ranking.

The second problem, addressed in Chapter 4, is to synthesize an "executive briefing" of a document. This task also has wide potential applicability. For instance, such a system could enable users of handheld information devices to absorb the information contained in large text documents more conveniently, despite the device's limited display capabilities. Chapter 4 describes a prototype system, called OCELOT, whose guiding philosophy differs from the prevailing one in automatic text summarization: rather than extracting a group of representative phrases and sentences from the document, OCELOT synthesizes an entirely new gist of the document, quite possibly with words not appearing in the original document. This "gisting" algorithm relies on a set of statistical models—whose parameters OCELOT learns automatically from a large collection of human-summarized documents—to guide its choice of words and how to arrange these words in a summary. There exists little previous work in this area and essentially no authoritative standards for adjudicating quality in a gist. But based on the qualitative and quantitative assessments appearing in Chapter 4, the results of this approach appear promising.

The final problem, which appears in Chapter 5, is in some sense a hybrid of the first two: succinctly characterize (or summarize) the relevance of a document to a query. For example, part of a newspaper article on skin care may be relevant to a teenager interested in handling an acne problem, while another part is relevant to someone older, more worried about wrinkles. The system described in Chapter 5 adapts to a user's information need in generating a *query-relevant summary*. Learning parameter values for the proposed model requires a large collection of summarized documents, which is difficult to obtain, but as a proxy, one can use a collection of FAQ (frequently-asked question) documents.

1.2 Learning to process text

Pick up any introductory book on algorithms and you'll discover, in explicit detail, how to program a computer to calculate the greatest common divisor of two numbers and to sort a list of names alphabetically. These are tasks which are easy to specify algorithmically.

This thesis is concerned with a set of language-related tasks that humans can perform, but which are difficult to specify algorithmically. For instance, it appears quite difficult to devise an automatic procedure for deciding if a body of text addresses the question `''How many kinds of mammals are bipedal?''`. Though this is a relatively straightforward task for a native English speaker, no one has yet invented a reliable algorithmic specification for it. One might well ask what such a specification would even look like. Adjudicating relevance based on whether the document contained key terms like `mammals` and `bipedal` won't do the trick: many documents containing both words have nothing whatsoever to do with the question. The converse is also true: a document may contain neither the word `mammals` nor the word `bipedal`, and yet still answer the question.

The following chapters describe how a computer can "learn" to perform rather sophisticated tasks involving natural language, by observing how a person performs the same task. The specific tasks addressed in the thesis are varied—ranking documents by relevance to a query, producing a gist of a document, and summarizing a document with respect to a topic. But a single strategy prevails throughout:

1. *Data collection*: Start with a large sample of data representing how humans perform the task.

2. *Model selection*: Settle on a parametric statistical model of the process.

3. *Parameter estimation*: Calculate parameter values for the model by inspection of the data.

Together, these three steps comprise the *construction* of the text processing system. The fourth step involves the *application* of the resulting system:

4. *Search*: Using the learned model, find the optimal solution to the given problem—the best summary of a document, for instance, or the document most relevant to a query, or the section of a document most germane to a user's information need.

There's a name for this approach—it's called statistical machine learning. The technique has been applied with success to the related areas of speech recognition, text classification, automatic language translation, and many others. This thesis represents a unified treatment using statistical machine learning of a wide range of problems in the field of information retrieval.

There's an old saying that goes something like "computers only do what people tell them to do." While strictly true, this saying suggests a overly-limited view of the power of automation. With the right tools, a computer can learn to perform sophisticated text-related tasks without being told explicitly how to do so.

1.3 Statistical machine learning for information retrieval

Before proceeding further, it seems appropriate to deconstruct the title of this thesis: *Statistical Machine Learning for Information Retrieval.*

Machine Learning

Machine Learning is, according to a recent textbook on the subject, "the study of algorithms which improve from experience" [62]. Machine learning is a rather diffuse field of inquiry, encompassing such areas as **reinforcement learning** (where a system, like a chess-playing program, improves its performance over time by favoring behavior resulting in a positive outcome), **online learning** (where a system, like an automatic stock-portfolio manager, optimizes its behavior while performing the task, by taking note of its performance so far) and **concept learning** (where a system continuously refines the set of viable solutions by eliminating those inconsistent with evidence presented thus far).

This thesis will take a rather specific view of machine learning. In these pages, the phrase "machine learning" refers to a kind of generalized regression: characterizing a set of labeled events $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$ with a function $\Phi : X \to Y$ from event to label (or "output"). Researchers have used this paradigm in countless settings. In one, X represents a medical image of a working heart: Y represents a clinical diagnosis of the pathology, if any, of the heart [78]. In machine translation, which lies closer to the topic at hand, X represents a sequence of (say) French words and Y a putative English translation of this sequence [6]. Loosely speaking, then, the "machine learning" part of the title refers to the process by which a computer creates an internal representation of a labeled dataset in order to predict the output corresponding to a new event.

The question of how accurately a machine can learn to perform a labeling task is an important one: accuracy depends on the amount of labeled data, the expressiveness of the internal representation, and the inherent difficulty of the labeling problem itself. An entire subfield of machine learning called computational learning theory has evolved in the past several years to formalize such questions [46], and impose theoretic limits on what an algorithm can and can't do. The reader may wish to ruminate, for instance, over the setting in which X is a computer program and Y a boolean indicating whether the program halts on all inputs.

Statistical Machine Learning

Statistical machine learning is a flavor of machine learning distinguished by the fact that the internal representation is a statistical model, often parametrized by a set of probabilities. For illustration, consider the syntactic question of deciding whether the word `chair` is acting as a verb or a noun within a sentence. Most any English-speaking fifth-grader would have little difficulty with this problem. But how to program a computer to perform this task? Given a collection of sentences containing the word `chair` and, for each, a labeling `noun` or `verb`, one could invoke a number of machine learning approaches to construct an automatic "syntactic disambiguator" for the word `chair`. A rule-inferential technique would construct an internal representation consisting of a list of lemmae, perhaps comprising a decision tree. For instance, the tree might contain a rule along the lines "If the word preceding `chair` is `to`, then `chair` is a verb." A simple statistical machine learning representation might contain this rule as well, but now equipped with a probability: "If the word preceding `chair` is `to`, then with probability p `chair` is a verb."

Statistical machine learning dictates that the parameters of the internal representation—

the p in the above example, for instance—be calculated using a well-motivated criterion. Two popular criteria are maximum likelihood and maximum *a posteriori* estimation. Chapter 2 contains a treatment of the standard objective functions which this thesis relies on.

Information Retrieval

For the purposes of this thesis, the term *Information Retrieval* (IR) refers to any large-scale automatic processing of text. This definition seems to overburden these two words, which really ought only to refer to the *retrieval* of information, and not to its translation, summarization, and classification as well. This document is guilty only of perpetuating dubious terminology, not introducing it; the premier Information Retrieval conference (ACM SIGIR) traditionally covers a wide range of topics in text processing, including information filtering, compression, and summarization.

Despite the presence of mathematical formulae in the upcoming chapters, the spirit of this work is practically motivated: the end goal was to produce not theories in and of themselves, but working systems grounded in theory. Chapter 3 addresses one IR-based task, describing a system called WEAVER which ranks documents by relevance to a query. Chapter 4 addresses a second, describing a system called OCELOT for synthesizing a "gist" of an arbitrary web page. Chapter 5 addresses a third task, that of identifying the contiguous subset of a document most relevant to a query—which is one strategy for summarizing a document with respect to the query.

1.4 Why now is the time

For a number of reasons, much of the work comprising this thesis would not have been possible ten years ago.

Perhaps the most important recent development for statistical text processing is the growth of the Internet, which consists (as of this writing) of over a billion documents[1]. This collection of hypertext documents is a dataset like none ever before assembled, both in sheer size and also in its diversity of topics and language usage. The rate of growth of this dataset is astounding: the Internet Archive, a project devoted to "archiving" the contents of the Internet, has attempted, since 1996, to spool the text of publicly-available Web pages to disk: the archive is well over 10 terabytes large and currently growing by two terabytes per month [83].

[1] A billion, that is, according to an accounting which only considers *static* web pages. There are in fact an infinite number of dynamically-generated web pages.

That the Internet represents an incomparable knowledge base of language usage is well known. The question for researchers working in the intersection of machine learning and IR is how to make use of this resource in building practical natural language systems. One of the contributions of this thesis is its use of novel resources collected from the Internet to estimate the parameters of proposed statistical models. For example,

- Using frequently-asked question (FAQ) lists to build models for answer-finding and query-relevant summarization;
- Using server logs from a large commercial web portal to build a system for assessing document relevance;
- Using a collection of human-summarized web pages to construct a system for document gisting.

Some researchers have in the past few years begun to consider how to leverage the growing collection of digital, freely available information to produce better natural language processing systems. For example, Nie has investigated the discovery and use of a corpus of web page pairs—each pair consisting of the same page in two different languages—to learn a model of translation between the languages [64]. Resnick's STRAND project at the University of Maryland focuses more on the automatic discovery of such web page pairs [73].

Learning statistical models from large text databases can be quite resource-intensive. The machine use to conduct the experiments in this thesis[2] is a Sun Microsystems 266Mhz six-processor UltraSparc workstation with 1.5GB of physical memory. On this machine, some of the experiments reported in later chapters required days or even weeks to complete. But what takes three days on this machine would require three months on a machine of less recent vintage, and so the increase in computational resources permits experiments today that were impractical until recently. Looking ahead, statistical models of language will likely become more expressive and more accurate, because training these more complex models will be feasible with tomorrow's computational resources. One might say "What Moore's Law giveth, statistical models taketh away."

1.5 A motivating example

This section presents a case study in statistical text processing which highlights many of the themes prevalent in this work.

[2]and, for that matter, to typeset this document

From a sequence of words $\mathbf{w} = \{w_1, w_2, \ldots w_n\}$, the part-of-speech labeling problem is to discover an appropriate set of syntactic labels `s`, one for each of the n words. This is a generalization of the "noun or verb?" example given earlier in this chapter. For instance, an appropriate labeling for `the quick brown fox jumped over the lazy dog` might be

w:	The	quick	brown	fox	jumped	over	the	lazy	dog	.
s:	DET	ADJ	ADJ	NOUN-S	VERB-P	PREP	DET	ADJ	NOUN-S	PUNC

A reasonable line of attack for this problem is to try to encapsulate into an algorithm the expert knowledge brought to bear on this problem by a linguist—or, even less ambitiously, an elementary school child. To start, it's probably safe to say that the word `the` just about always behaves as a determiner (`DET` in the above notation); but after picking off this and some other low-hanging fruit, hope of specifying the requisite knowledge quickly fades. After all, even a word like `dog` could, in some circumstances, behave as a verb[3]. Because of this difficulty, the earliest automatic tagging systems, based on an expert-systems architecture, achieved a per-word accuracy of only around 77% on the popular Brown corpus of written English [37].

(The Brown corpus is a 1,014,312-word corpus of running English text excerpted from publications in the United States in the early 1960's. For years, the corpus has been a popular benchmark for assessing the performance of general natural-language algorithms [30]. The reported number, 77%, refers to the accuracy of the system on an "evaluation" portion of the dataset, not used during the construction of the tagger.).

Surprisingly, perhaps, it turns out that a knowledge of English syntax isn't at all necessary—or even helpful—in designing an accurate tagging system. Starting with a collection of text in which each word is annotated with its part of speech, one can apply statistical machine learning to construct an accurate tagger. A successful architecture for a statistical part of speech tagger uses Hidden Markov Models (HMMs), an abstract state machine whose states are different parts of speech, and whose output symbols are words. In producing a sequence of words, the machine progresses through a sequence of states corresponding to the parts of speech for these words, and at each state transition outputs the next word in the sequence. HMMs are described in detail in Chapter 2.

It's not entirely clear who was first responsible for the notion of applying HMMs to the part-of-speech annotation problem; much of the earliest research involving natural language processing and HMMs was conducted behind a veil of secrecy at defense-related U.S. government agencies. However, the earliest account in the scientific literature appears to be Bahl and Mercer in 1976 [4].

[3] And come to think of it, in a pathological example, so could "`the`."

Conveniently, there exist several part-of-speech-annotated text corpora, including the Penn Treebank, a $43,113$-sentence subset of the Brown corpus [57]. After automatically learning model parameters from this dataset, HMM-based taggers have achieved accuracies in the 95% range [60].

This example serves to highlight a number of concepts which will appear again and again in these pages:

- *The viability of probabilistic methods*: Most importantly, the success of Hidden Markov Model tagging is a substantiation of the claim that knowledge-free (in the sense of not explicitly embedding any expert advice concerning the target problem) probabilistic methods are up to the task of sophisticated text processing—and, more surprisingly, can outperform knowledge-rich techniques.

- *Starting with the right dataset*: In order to learn a pattern of intelligent behavior, a machine learning algorithm requires examples of the behavior. In this case, the Penn Treebank provides the examples, and the quality of the tagger learned from this dataset is only as good as the dataset itself. This is a restatement of the first part of the four-part strategy outlined at the beginning of this chapter.

- *Intelligent model selection*: Having a high-quality dataset from which to learn a behavior does not guarantee success. Just as important is discovering the right statistical model of the process—the second of our four-part strategy. The HMM framework for part of speech tagging, for instance, is rather non-intuitive. There are certainly many other plausible models for tagging (including exponential models [72], another technique relying on statistical learning methods), but none so far have proven demonstrably superior to the HMM approach.

  Statistical machine learning can sometimes feel formulaic: postulate a parametric form, use maximum likelihood and a large corpus to estimate optimal parameter values, and then apply the resulting model. The science is in the parameter estimation, but the art is in devising an expressive statistical model of the process whose parameters can be feasibly and robustly estimated.

1.6 Foundational work

There are two types of scientific precedent for this thesis. First is the slew of recent, related work in statistical machine learning and IR. The following chapters include, whenever appropriate, reference to these precedents in the literature. Second is a small body of seminal work which lays the foundation for the work described here.

Information theory is concerned with the production and transmission of information. Using a framework known as the source-channel model of communication, information theory has established theoretical bounds on the limits of data compression and communication in the presence of noise and has contributed to practical technologies as varied as cellular communication and automatic speech transcription [2, 22]. Claude Shannon is generally credited as having founded the field of study with the publication in 1948 of an article titled "A mathematical theory of communication," which introduced the notion of measuring the entropy and information of random variables [79]. Shannon was also as responsible as anyone for establishing the field of statistical text processing: his 1951 paper "Prediction and Entropy of Printed English" connected the mathematical notions of entropy and information to the processing of natural language [80].

From Shannon's explorations into the statistical properties of natural language arose the idea of a language model, a probability distribution over sequences of words. Formally, a language model is a mapping from sequences of words to the portion of the real line between zero and one, inclusive, in which the total assigned probability is one. In practice, text processing systems employ a language model to distinguish likely from unlikely sequences of words: a useful language model will assign a higher probability to `A bird in the hand` than `hand the a bird in`. Language models form an integral part of modern speech and optical character recognition systems [42, 63], and in information retrieval as well: Chapter 3 will explain how the WEAVER system can be viewed as a generalized type of language model, Chapter 4 introduces a gisting prototype which relies integrally on language-modelling techniques, and Chapter 5 uses language models to rank candidate excerpts of a document by relevance to a query.

Markov Models were invented by the Russian mathematician A. A. Markov in the early years of the twentieth century as a way to represent the statistical dependencies among a set of random variables. An abstract state machine is *Markovian* if the state of the machine at time $t+1$ and time $t-1$ are conditionally independent, given the state at time t. The application Markov had in mind was, perhaps coincidentally, related to natural language: modeling the vowel-consonant structure of Russian [41]. But the tools he developed had a much broader import and subsequently gave rise to the study of stochastic processes.

Hidden Markov Models are a statistical tool originally designed for use in robust digital transmission and subsequently applied to a wide range of problems involving pattern recognition, from genome analysis to optical character recognition [26, 54]. A discrete Hidden Markov Model (HMM) is an automaton which moves between a set of states and produces, at each state, an output symbol from a finite vocabulary. In general, both the movement between states and the generated symbols are probabilistic, governed by the values in a stochastic matrix.

Applying HMMs to text and speech processing started to gain popularity in the 1970's, and a 1980 symposium sponsored by the Institute for Defense Analysis contains a number of important early contributions. The editor of the papers collected from that symposium, John Ferguson, wrote in a preface that

> *Measurements of the entropy of ordinary Markov models for language reveal that a substantial portion of the inherent structure of the language is not included in the model. There are also heuristic arguments against the possibility of capturing this structure using Markov models alone.*
>
> *In an attempt to produce stronger, more efficient models, we consider the notion of a* Hidden Markov model. *The idea is a natural generalization of the idea of a Markov model...This idea allows a wide scope of ingenuity in selecting the structure of the states, and the nature of the probabilistic mapping. Moreover, the mathematics is not hard, and the arithmetic is easy, given access to a modern computer.*

The "ingenuity" to which the author of the above quotation refers is what Section 1.2 labels as the second task: model selection.

Chapter 2

Mathematical machinery

This chapter reviews the mathematical tools on which the following chapters rely: rudimentary information theory, maximum likelihood estimation, convexity, the EM algorithm, mixture models and Hidden Markov Models.

The statistical modelling problem is to characterize the behavior of a real or imaginary stochastic process. The phrase *stochastic process* refers to a machine which generates a sequence of output values or "observations" Y: pixels in an image, horse race winners, or words in text. In the language-based setting we're concerned with, these values typically correspond to a discrete time series.

The modelling problem is to come up with an accurate (in a sense made precise later) model λ of the process. This model assigns a probability $p_\lambda(Y = y)$ to the event that the random variable Y takes on the value y. If the identity of Y is influenced by some conditioning information X, then one might instead seek a conditional model $p_\lambda(Y \mid X)$, assigning a probability to the event that symbol y appears within the context x.

The *language modelling problem*, for instance, is to construct a conditional probability distribution function (p.d.f.) $p_\lambda(Y \mid X)$, where Y is the identity of the next word in some text, and X is the conditioning information, such as the identity of the preceding words. Machine translation [6], word-sense disambiguation [10], part-of-speech tagging [60] and parsing of natural language [11] are just a few other human language-related domains involving stochastic modelling.

Before beginning in earnest, a few words on notation are in place. In this thesis (as in almost all language-processing settings) the random variables Y are discrete, taking on values in some finite alphabet $\mathcal{Y}$—a vocabulary of words, for example. Heeding convention, we will denote a specific value taken by a random variable Y as y.

For the sake of simplicity, the notation in this thesis will sometimes obscure the distinction between a random variable Y and a value y taken by that random variable. That is, $p_\lambda(Y = y)$ will often be shortened to $p_\lambda(y)$. Lightening the notational burden even further, $p_\lambda(y)$ will appear as $p(y)$ when the dependence on λ is entirely clear. When necessary to distinguish between a single word and a vector (e.g. phrase, sentence, document) of words, this thesis will use bold symbols to represent word vectors: s is a single word, but $\mathbf{s}$ is a sentence.

2.1 Building blocks

One of the central topics of this chapter is the EM algorithm, a hill-climbing procedure for discovering a locally optimal member of a parametric family of models involving hidden state. Before coming to this algorithm and some of its applications, it makes sense to introduce some of the major players: entropy and mutual information, maximum likelihood, convexity, and auxiliary functions.

2.1.1 Information theory

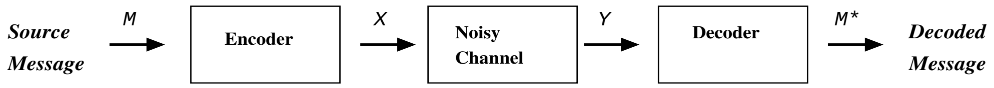


Figure 2.1: The source-channel model in information theory

The field of information theory, as old as the digital computer, concerns itself with the efficient, reliable transmission of information. Figure 2.1 depicts the standard information theoretic view of communication. In some sense, information theory is the study of what occurs in the boxes in this diagram.

Encoding: Before transmitting some message M across an unreliable channel, the sender may add redundancy to it, so that noise introduced in transit can be identified and corrected by the receiver. This is known as *error-correcting coding*. We represent encoding by a function $\psi : M \to X$.

Channel: Information theorists have proposed many different ways to model how information is compromised in traveling through a channel. A "channel" is an abstraction for a telephone wire, Ethernet cable, or any other medium (including time) across which a message can become corrupted. One common characterization of a channel is to imagine that it acts independently on each input sent through it. Assuming this "memoryless" property, the channel may be characterized by a conditional probability distribution $p(Y \mid X)$, where X is a random variable representing the input to the channel, and Y the output.

Decoding: The inverse of encoding: given a message M which was encoded into $\psi(M)$ and then corrupted via $p(Y \mid \psi(M))$, recover the original message. Assuming the source emits messages according to some known distribution $p(M)$, decoding amounts to finding

$$\begin{aligned} m^{\star} &= \arg\max_{m} p(\psi(m) \mid y) \\ &= \arg\max_{m} p(y \mid \psi(m))\, p(m), \end{aligned} \tag{2.1}$$

where the second equality follows from Bayes' Law.

To the uninitiated, (2.1) might appear a little strange. The goal is to discover the optimal message $m^{\star}$, but (2.1) suggests doing so by generating (or "predicting") the input Y. Far more than a simple application of Bayes' law, there are compelling reasons why the ritual of turning a search problem around to predict the input should be productive. When designing a statistical model for language processing tasks, often the most natural route is to build a *generative* model which builds the output step-by-step. Yet to be effective, such models need to liberally distribute probability mass over a huge number of possible outcomes. This probability can be difficult to control, making an accurate direct model of the distribution of interest difficult to fashion. Time and again, researchers have found that predicting what is already known from competing hypotheses is easier than directly predicting all of the hypotheses.

One classical application of information theory is communication between source and receiver separated by some distance. Deep-space probes and digital wireless phones, for example, both use a form of codes based on polynomial arithmetic in a finite field to guard against losses and errors in transmission. Error-correcting codes are also becoming popular for guarding against packet loss in Internet traffic, where the technique is known as *forward error correction* [33].

The source-channel framework has also found application in settings seemingly unrelated to communication. For instance, the now-standard approach to automatic speech recognition views the problem of transcribing a human utterance from a source-channel perspective [3]. In this case, the source message is a sequence of words M. In contrast to communication via error-correcting codes, we aren't free to select the code here—rather, it's the product of thousands of years of linguistic evolution. The encoding function maps a sequence of words to a pronunciation X, and the channel "corrupts" this into an acoustic signal Y—in other words, the sound emitted by the person speaking. The decoder's responsibility is to recover the original word sequence M, given

- the received acoustic signal Y,
- a model $p(Y \mid X)$ of how words sound when voiced,
- a prior distribution $p(X)$ over word sequences, assigning a higher weight to more fluent sequences and lower weight to less fluent sequences.

One can also apply the source-channel model to language translation. Imagine that the person generating the text to be translated originally thought of a string X of English words, but the words were "corrupted" into a French sequence Y in writing them down. Here again the channel is purely conceptual, but no matter; decoding is still a well-defined problem of recovering the original English x, given the observed French sequence Y, a model $p(Y \mid X)$ for how English translates to French, and a prior $p(X)$ on English word sequences [6].

2.1.2 Maximum likelihood estimation

Given is some observed sample $\mathbf{s} = \{s_1, s_2, \ldots s_N\}$ of the stochastic process. Fix an unconditional model λ assigning a probability $p_\lambda(S = \mathbf{s})$ to the event that the process emits the symbol $\mathbf{s}$. (A model is called *unconditional* if its probability estimate for the next emitted symbol is independent of previously emitted symbols.) The probability (or *likelihood*) of the sample $\mathbf{s}$ with respect to λ is

$$p(\mathbf{s} \mid \lambda) = \prod_{i=1}^{N} p_\lambda(S = s_i) \tag{2.2}$$

Equivalently, denoting by $c(y)$ the number of occurrences of symbol y in $\mathbf{s}$, we can rewrite the likelihood of $\mathbf{s}$ as

$$p(\mathbf{s} \mid \lambda) = \prod_{y \in \mathcal{Y}} p_\lambda(y)^{c(y)} \tag{2.3}$$

Within some prescribed family $\mathcal{F}$ of models, the *maximum likelihood* model is that λ assigning the highest probability to $\mathbf{s}$:

$$\lambda^\star \equiv \arg\max_{\lambda \in \mathcal{F}} p(\mathbf{s} \mid \lambda) \tag{2.4}$$

The likelihood is monotonically related to the average per-symbol log-likelihood,

$$L(\mathbf{s} \mid \lambda) \equiv \log p(\mathbf{s} \mid \lambda) = \sum_{y \in \mathcal{Y}} \frac{c(y)}{N} \log p_\lambda(y), \tag{2.5}$$

So the maximum likelihood model $\lambda^\star = \arg\max_{\lambda \in \mathcal{F}} L(\mathbf{s} \mid \lambda)$. Since it's often more convenient mathematically, it makes sense in practice to work in the log domain when searching for the maximum likelihood model.

The per-symbol log-likelihood has a convenient information theoretic interpretation. If two parties use the model λ to encode symbols—optimally assigning shorter codewords to symbols with higher probability and vice versa—then the per-symbol log-likelihood is the average number of bits per symbol required to communicate $\mathbf{s} = \{s_1, s_2 \ldots s_N\}$. And the average per-symbol *perplexity* of $\mathbf{s}$, a somewhat less popular metric, is related by $2^{-L(\mathbf{s}|\lambda)}$ [2, 48].

The maximum likelihood criterion has a number of desirable theoretical properties [17], but its popularity is largely due to its empirical success in selected applications and in the convenient algorithms it gives rise to, like the EM algorithm. Still, there are reasons *not* to rely overly on maximum likelihood for parameter estimation. After all, the sample of observed output which constitutes $\mathbf{s}$ is only a representative of the process being modelled. A procedure which optimizes parameters based on this sample alone—as maximum likelihood does—is liable to suffer from overfitting. Correcting an overfitted model requires techniques such as smoothing the model parameters using some data held out from the training [43, 45]. There have been many efforts to introduce alternative parameter-estimation approaches which avoid the overfitting problem during training [9, 12, 82].

Some of these alternative approaches, it turns out, are not far removed from maximum likelihood. *Maximum a posteriori* (MAP) modelling, for instance, is a generalization of maximum likelihood estimation which aims to find the most likely model given the data:

$$\lambda^\star = \arg\max_\lambda p(\lambda \mid \mathbf{s})$$

Using Bayes' rule, the MAP model turns out to be the product of a prior term and a likelihood term:

$$\lambda^\star = \arg\max_\lambda p(\lambda) p(\mathbf{s} \mid \lambda)$$

If one takes $p(\lambda)$ to be uniform over all λ, meaning that all models λ are *a priori* equally probable, MAP and maximum likelihood are equivalent.

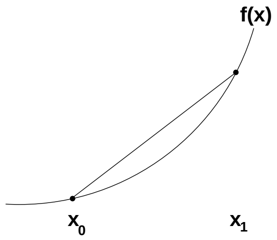
A slightly more interesting use of the prior $p(\lambda)$ would be to rule out (by assigning $p(\lambda) = 0$) any model λ which itself assigns zero probability to any event (that is, any model on the boundary of the simplex, whose support is not the entire set of events).

2.1.3 Convexity

A function $f(x)$ is *convex* ("concave up") if

$$f(\alpha x_0 + (1-\alpha)x_1) \leq \alpha f(x_0) + (1-\alpha) f(x_1) \quad \text{for all } 0 \leq \alpha \leq 1. \tag{2.6}$$

That is, if one selects any two points x_0 and x_1 in the domain of a convex function, the function always lies on or under the chord connecting x_0 and x_1:



A sufficient condition for convexity—the one taught in high school calculus—is that f is convex if and only if $f''(x) \geq 0$. But this is not a necessary condition, since f may not be everywhere differentiable; (2.6) is preferable because it applies even to non-differentiable functions, such as $f(x) = | x |$ at $x = 0$.

A multivariate function may be convex in any number of its arguments.

2.1.4 Jensen's inequality

Among the most useful properties of convex functions is that if f is convex in x, then

$$f(E[x]) \leq E[f(x)] \quad \text{or} \quad f(\sum_x p(x)x) \leq \sum_x p(x)f(x) \tag{2.7}$$

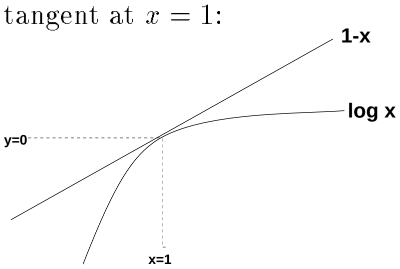
where $p(x)$ is a p.d.f. This follows from (2.6) by a simple inductive proof.

What this means, for example, is that (for any p.d.f. p) the following two conditions hold:

$$\sum_x p(x) \log f(x) \geq \log \sum_x p(x)f(x) \quad \text{since } -\log \text{ is convex} \tag{2.8}$$

$$\exp \sum_x p(x)f(x) \leq \sum_x p(x) \exp f(x) \quad \text{since exp is convex} \tag{2.9}$$

We'll also find use for the fact that a concave function always lies below its tangent; in particular, $\log x$ lies below its tangent at $x = 1$:



2.1.5 Auxiliary functions

At their most general, auxiliary functions are simply pointwise lower (or upper) bounds on a function. We've already seen an example: $x - 1$ is an auxiliary function for $\log x$ in the sense that $x - 1 \geq \log x$ for all x. This observation might prove useful if we're trying to establish that some function $f(x)$ lies on or above $\log x$: if we can show $f(x)$ lies on or above $x - 1$, then we're done, since $x - 1$ itself lies above $\log x$. (Incidentally, it's also true that $\log x$ is an auxiliary function for $x - 1$, albeit in the other direction).

We'll be making use of a particular type of auxiliary function: one that bounds the change in log-likelihood between two models. If λ is one model and λ' another, then we'll be interested in the quantity $L(s \mid \lambda') - L(s \mid \lambda)$, the *gain* in log-likelihood from using λ' instead of λ. For the remainder of this chapter, we'll define $A(\lambda', \lambda)$ to be an auxiliary function only if

$$L(s \mid \lambda') - L(s \mid \lambda) \geq A(\lambda', \lambda) \quad \text{and} \quad A(\lambda, \lambda) = 0$$

Together, these conditions imply that if we can find a λ' such that $A(\lambda', \lambda) > 0$, then λ' is a better model than λ—in a maximum likelihood sense.

The core idea of the EM algorithm, introduced in the next section, is to iterate this process in a hill-climbing scheme. That is, start with some model λ, replace λ by a superior model λ', and repeat this process until no superior model can be found; in other words, until reaching a stationary point of the likelihood function.

2.2 EM algorithm

The standard setting for the EM algorithm is as follows. The stochastic process in question emits observable output Y (words for instance), but this data is an *incomplete* representation of the process. The complete data will be denoted by (Y, H)—H for "partly hidden." Focusing on the discrete case, we can write y_i as the observed output at time i, and h_i as the state of the process at time i.

The EM algorithm is an especially convenient tool for handling Hidden Markov Models (HMMs). HMMs are a generalization of traditional Markov models: whereas each state-to-state transition on a Markov model causes a specific symbol to be emitted, each state-state transition on an HMM contains a *probability distribution* over possible emitted symbols. One can think of the state as the hidden information and the emitted symbol as the observed output. For example, in an HMM part-of-speech model, the observable data are the words and the hidden states are the parts of speech.

The EM algorithm arises in other human-language settings as well. In a parsing model, the words are again the observed output, but now the hidden state is the parse of the sentence [53]. Some recent work on statistical translation (which we will have occasion to revisit later in this thesis) describes an English-French translation model in which the alignment between the words in the French sentence and its translation represents the hidden information [6].

We postulate a parametric model $p_\lambda(Y, H)$ of the process, with marginal distribution $p_\lambda(Y) = \sum_h p_\lambda(Y, H = h)$. Given some empirical sample $\mathbf{s}$, the principle of maximum likelihood dictates that we find the λ which maximizes the likelihood of $\mathbf{s}$. The difference in log-likelihood between models λ' and λ is

$$\begin{aligned} L(\mathbf{s} \mid \lambda') - L(\mathbf{s} \mid \lambda) &= \sum_y q(y) \log \frac{p_{\lambda'}(y)}{p_\lambda(y)} \\ &= \sum_y q(y) \log \sum_h \frac{p_{\lambda'}(y,h)}{p_\lambda(y)} \\ &= \sum_y q(y) \log \sum_h \frac{p_{\lambda'}(y,h) p_\lambda(h \mid y)}{p_\lambda(y,h)} \qquad \text{applying Bayes' law to } p_\lambda(y) \\ &\geq \underbrace{\sum_y q(y) \sum_h p_\lambda(h \mid y) \log \frac{p_{\lambda'}(y,h)}{p_\lambda(y,h)}}_{\text{Call this } Q(\lambda' \mid \lambda)} \qquad \text{applying (2.8)} \end{aligned} \tag{2.10}$$

We've established that $L(s \mid \lambda') - L(s \mid \lambda) \geq Q(\lambda' \mid \lambda)$. It's also true (by inspection) that $Q(\lambda \mid \lambda) = 0$. Together, these earn Q the title of auxiliary function to L. If we can find a λ' for which $Q(\lambda' \mid \lambda) > 0$, then $p_{\lambda'}$ has a higher (log)-likelihood than p_λ.

This observation is the basis of the EM (expectation-maximization) algorithm.

Algorithm 1: *Expectation-Maximization (EM)*

1. (*Initialization*) `Pick a starting model` λ

2. `Repeat until log-likelihood convergences:`

   (*E-step*) `Compute` $Q(\lambda' \mid \lambda)$

   (*M-step*) $\lambda \leftarrow \arg\max_{\lambda'} Q(\lambda' \mid \lambda)$

A few points are in order about the algorithm.

- The algorithm is greedy, insofar as it attempts to take the best step from the current λ at each iteration, paying no heed to the global behavior of $L(s \mid \lambda)$. The line of reasoning culminating in (2.10) established that each step of the EM algorithm can never produce an inferior model. But this doesn't rule out the possibility of

- Getting "stuck" at a local maximum
- Toggling between two local maxima corresponding to different models with identical likelihoods.

Denoting by λ_i the model at the ith iteration of Algorithm 1, under certain assumptions it can be shown that $\lim_n \lambda^n = \lambda^\star$. That is, eventually the EM algorithm converges to the optimal parameter values [88]. Unfortunately, these assumptions are rather restrictive and aren't typically met in practice.

It may very well happen that the space is very "bumpy," with lots of local maxima. In this case, the result of the EM algorithm depends on the starting value λ_0; the algorithm might very well end up at a local maximum. One can enlist any number of heuristics for high-dimensional search in an effort to find the global maximum, such as selecting a number of different starting points, searching by simulating annealing, and so on.

- Along the same line, if each iteration is computationally expensive, it can sometimes pay to try to speed convergence by using second-derivative information. This technique is known variously as Aitken's acceleration algorithm or "stretching" [1]. However, this technique is often unviable because Q'' is hard to compute.

- In certain settings it can be difficult to maximize $Q(\lambda' \mid \lambda)$, but rather easy to find *some* λ' for which $Q(\lambda' \mid \lambda) > 0$. But that's just fine: picking this λ' still improves the likelihood, though the algorithm is no longer greedy and may well run slower. This version of the algorithm—replacing the "M"-step of the algorithm with some technique for simply taking a step in the right direction, rather than the maximal step in the right direction—is known as the GEM algorithm (G for "generalized").

2.2.1 Example: mixture weight estimation

A quite typical problem in statistical modelling is to construct a mixture model which is the linear interpolation of a collection of models. We start with an observed sample of output $\{y_1, y_2, \ldots y_T\}$ and a collection of distributions $p_1(y), p_2(y) \ldots p_N(y)$. We seek the maximum likelihood member of the family of distributions

$$\mathcal{F} \equiv \left\{ p(Y = y) = \sum_{i=1}^{N} \alpha_i p_i(y) \mid \alpha_i \geq 0 \text{ and } \sum_i \alpha_i = 1 \right\}$$

Members of $\mathcal{F}$ are just linear interpolations—or "mixture models"—of the individual models p_i, with different members distributing their weights differently across the models. The problem is to find the best mixture model. On the face of it, this appears to be an $(N-1)$-dimensional search problem. But the problem yields quite easily to an EM approach.

Imagine the interpolated model is at any time in one of N states, $a \in \{1, 2, \ldots N\}$, with:

- α_i: the *a priori* probability that the model is in state i at some time;
- $p_\lambda(a = i, y) = \alpha_i p_i(y)$: the probability of being in state i and producing output y;
- $p_\lambda(a = i \mid y) = \dfrac{\alpha_i p_i(y)}{\sum_i \alpha_i p_i(y)}$: the probability of being in state i, *given* that y is the current output

A convenient way to think of this is that in state i, the interpolated model relies on the i'th model. The appropriate version of (2.10) is, in this case,

$$Q(\alpha' \mid \alpha) = \sum_y q(y) \sum_{a=1}^{N} p_\lambda(a \mid y) \log \frac{p_{\lambda'}(y, a)}{p_\lambda(y, a)}$$

The EM algorithm says to find the α' maximizing $Q(\alpha' \mid \alpha)$—subject, in this case, to $\sum_i \alpha'_i = 1$. Applying the method of Lagrange multipliers,

$$\begin{aligned}
\frac{\partial}{\partial \alpha'_i} \left[Q(\alpha' \mid \alpha) - \gamma(\sum_i \alpha'_i - 1) \right] &= 0 \\
\sum_y q(y) p_\lambda(a = i \mid y) \frac{1}{p_{\lambda'}(y, a = i)} p_i(y) - \gamma &= 0 \\
\sum_y q(y) p_\lambda(a = i \mid y) \frac{1}{\alpha'_i} - \gamma &= 0
\end{aligned}$$

To ease the notational burden, introduce the shorthand

$$\begin{aligned}
C_i &\equiv \sum_y q(y) p_\lambda(a = i \mid y) \frac{1}{\alpha'_i} \\
&= \frac{1}{\alpha'_i} \sum_y q(y) \frac{\alpha_i p_i(y)}{\sum_i \alpha_i p_i(y)}
\end{aligned} \tag{2.11}$$

Applying the normalization constraint gives $\alpha'_i = \frac{C_i}{\sum_i C_i}$. Intuitively, C_i is the expected number of times the i'th model is used in generating the observed sample, given the current estimates for $\{\alpha_1, \alpha_2, \ldots \alpha_n\}$.

This is, once you think about it, quite an intuitive approach to the problem. Since we don't know the linear interpolation weights, we'll guess them, apply the interpolated model to

Algorithm 2: *EM for calculating mixture model weights*

1. (*Initialization*) `Pick initial weights` α `such that` $\alpha_i \in (0,1)$ `for all` i

2. `Repeat until convergence:`

   (*E-step*) `Compute` $C_1, C_2, \ldots C_N$`, given the current` α`, using (2.11).`

   (*M-step*) `Set` $\alpha_i \leftarrow \frac{C_i}{\sum_i C_i}$

---

the data, and see how much each individual model contributes to the overall prediction. Then we can update the weights to favor the models which had a better track record, and iterate. It's not difficult to imagine that someone might think up this algorithm without having the mathematical equipment (in the EM algorithm) to prove anything about it. In fact, at least two people did [39] [86].

* * *

A practical issue concerning the EM algorithm is that the sum over the hidden states H in computing (2.10) can, in practice, be an exponential sum. For instance, the hidden state might represent part-of-speech labelings for a sentence. If there exist T different part of speech labels, then a sentence of length n has T^n possible labelings, and thus the sum is over T^n hidden states. Often some cleverness suffices to sidestep this computational hurdle—usually by relying on some underlying Markov property of the model. Such cleverness is what distinguishes the Baum-Welch or "forward-backward" algorithm. Chapters 3 and 4 will face these problems, and wil use a combinatorial sleight of hand to calculate the sum efficiently.

2.3 Hidden Markov Models

Recall that a stochastic process is a machine which generates a sequence of output values $\mathbf{o} = \{o_1, o_2, o_3 \ldots o_n\}$, and a stochastic process is called Markovian if the state of the machine at time $t+1$ and at time $t-1$ are conditionally independent, given the state at time t:

$$p(o_{t+1} \mid o_{t-1} o_t) = p(o_{t+1} \mid o_t) \quad \text{and} \quad p(o_{t-1} \mid o_t o_{t+1}) = p(o_{t-1} \mid o_t)$$

In other words, the past and future observations are independent, given the present observation. A Markov Model may be thought of as a graphical method for representing this statistical independence property.

A Markov model with n states is characterized by n^2 transition probabilities $p(i, j)$—the probability that the model will move to state j from state i. Given an observed state sequence, say the state of an elevator at each time interval,

o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}	o_{11}
1st	*1st*	*2nd*	*3rd*	*3rd*	*2nd*	*2nd*	*1st*	*stalled*	*stalled*	*stalled*

one can calculate the maximum likelihood values for each entry in this matrix simply by counting: $p(i, j)$ is the number of times state j followed state i, divided by the number of times state i appeared before *any* state.

Hidden Markov Models (HMMs) are a generalization of Markov Models: whereas in conventional Markov Models the state of the machine at time i and the observed output at time i are one and the same, in Hidden Markov Models the state and output are decoupled. More specifically, in an HMM the automaton generates a symbol probabilistically at each state; only the symbol, and not the identity of the underlying state, is visible.

To illustrate, imagine that a person is given a newspaper and is asked to classify the articles in the paper as belonging to either the business section, the weather, sports, horoscope, or politics. At first the person begins reading an article which happens to contain the words `shares`, `bank`, `investors`; in all likelihood their eyes have settled on a business article. They next flip the pages and begin reading an article containing the words `front` and `showers`, which is likely a weather article. Figure 2.2 shows an HMM corresponding to this process—the states correspond to the categories, and the symbols output from each state correspond to the words in articles from that category. According to the values in the figure, the word `taxes` accounts for 2.2 percent of the words in the **news** category, and 1.62 percent of the words in the **business** category. Seeing the word `taxes` in an article does not by itself determine the most appropriate labeling for the article.

To fully specify an HMM requires four ingredients:

- The number of states $\mid S \mid$
- The number of output symbols $\mid W \mid$
- The state-to-state transition matrix, consisting of $\mid S \mid \times \mid S \mid$ parameters
- An output distribution over symbols for each state: $\mid W \mid$ parameters for each of the $\mid S \mid$ states.

In total, this amounts to $S(S-1)$ free parameters for the transition probabilities, and $W-1$ free parameters for the output probabilities.

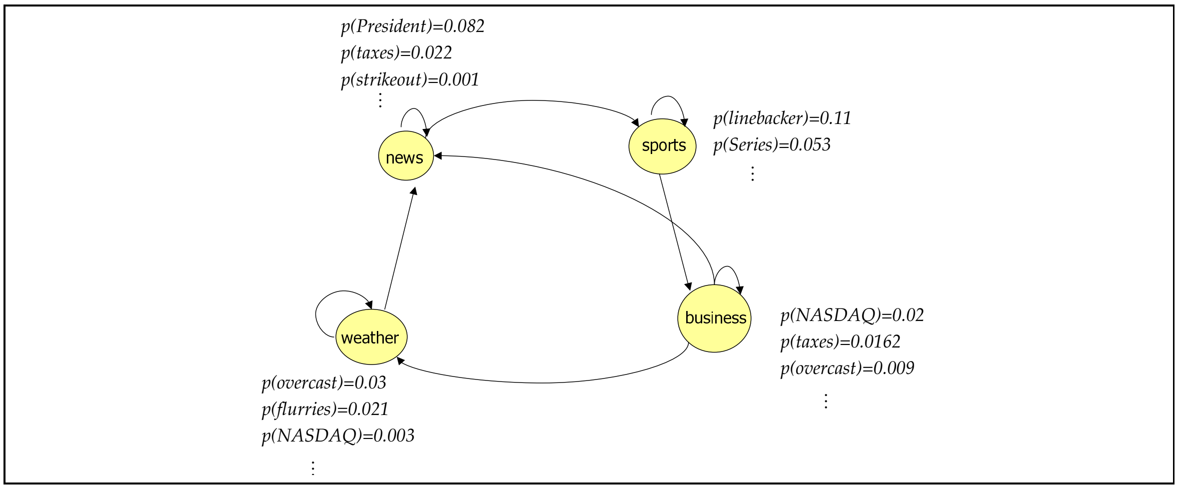


Figure 2.2: A Hidden Markov Model for text categorization.

2.3.1 Urns and mugs

Imagine an urn containing an unknown fraction $b(\circ)$ of white balls and a fraction $b(\bullet)$ of black balls. If in drawing T times with replacement from the urn we retrieve k white balls, then a plausible estimate for $b(\circ)$ is k/T. This is not only the intuitive estimate but also the maximum likelihood estimate, as the following line of reasoning establishes.

Setting $\gamma \equiv b(\circ)$, the probability of drawing $n = k$ white balls when sampling with replacement T times is

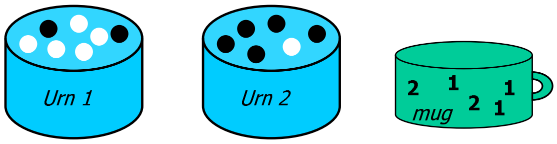
$$p(n = k) = \binom{T}{k} \gamma^k (1-\gamma)^{T-k}$$

The maximum likelihood value of γ is

$$\begin{aligned}
\arg\max_{\gamma} p(n = k) &= \arg\max_{\gamma} \binom{T}{k} \gamma^k (1-\gamma)^{T-k} \\
&= \arg\max_{\gamma} \left(\log \binom{T}{k} + k \log \gamma + (T-k) \log(1-\gamma) \right)
\end{aligned}$$

Differentiating with respect to γ and setting the result to zero yields $\gamma = k/T$, as expected.

Now we move to a more interesting scenario, directly relevant to Hidden Markov Models. Say we have two urns and a mug:



Denote:

$$\begin{aligned}
b_x(\circ) &= \text{fraction of white balls in urn } x \\
b_x(\bullet) &= \text{fraction of black balls in urn } x \ (= 1 - b_x(\circ)) \\
a_1 &= \text{fraction of 1's in mug} \\
a_2 &= \text{fraction of 2's in mug } (= 1 - a_1)
\end{aligned}$$

To generate a single output symbol using this model, we apply the following procedure: First, draw a number x from the mug; then draw a ball from urn x. This process represents a *mixture model*: the urns are states, and the black and white balls are outputs. The probability of drawing a single black ball is:

$$p(\bullet) = p(\text{urn } 1)p(\bullet \mid \text{urn } 1) + p(\text{urn } 2)p(\bullet \mid \text{urn } 2)$$

The process is also an HMM: the mug represents the hidden state and the balls represent the outputs. An output sequence consisting of white and black balls can arise from a large number of possible state sequences.

Algorithm 3: *EM for urn density estimation*

1. (*Initialization*) `Pick a starting value` $a_1 \in (0, 1)$
2. `Repeat until convergence:`

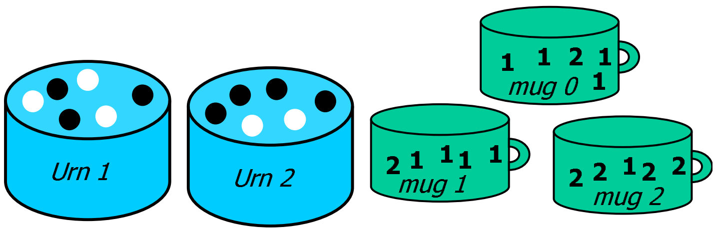
   (*E-step*) `Compute expected number of draws from urn` 1 `and` 2
   `in generating` $\mathbf{o}$`:` $c(1) \stackrel{\text{def}}{=} E[\# \text{ from urn } 1 \mid \mathbf{o}]$

   (*M-step*) $a_1 \leftarrow \dfrac{c(1)}{c(1) + c(2)}$

One important question which arises in working with models of this sort is to estimate maximum-likelihood values for the model parameters, given a sample $\mathbf{o} = \{o_1, o_2, \ldots o_T\}$ of output from the model. For simplicity, we'll restrict attention to calculating a maximum-likelihood estimate for a_1, the fraction of 1's in the mug. In this setting, one can think of the EM algorithm as a procedure of iteratively refining an estimate for a_1.

2.3.2 Three problems

The previous example was still too simple, since the underlying Markov chain was zeroth order, meaning that the probability of drawing a number from the urn is independent of the previously-drawn number. One can add "history" to the setup by introducing two more mugs:



The generative model corresponding to this setting is:

1. Draw a number x from mug 0
2. Draw a ball from urn x
3. Draw a new number $\hat{x}$ from mug x
4. Set $x \leftarrow \hat{x}$ and go to step 2.

We'll denote by a_{x1} the fraction of 1's in mug x; that is, the probability that after drawing from mug x, the next urn drawn from is 1.

The three canonical HMM-related problems are:

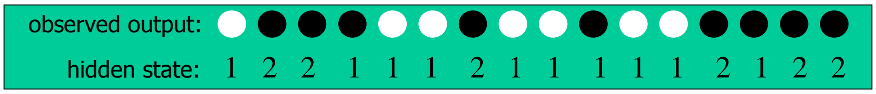
I. *Probability calculation*: What is the probability that an HMM with known parameters will generate $\mathbf{o} = \{o_1, o_2 \ldots o_T\}$ as output?

II. *Decoding*: What's the most likely state sequence that an HMM of known parameters followed in generating $\mathbf{o} = \{o_1, o_2 \ldots o_T\}$?

III. *Parameter estimation*: Given that an HMM with unknown parameters generated the output sequence $\mathbf{o} = \{o_1, o_2, \ldots o_T\}$, estimate maximum-likelihood values for the HMM's parameters.

I. Probability calculation

To begin, we'll address an easier version of the first problem: the joint probability of an output sequence *and* a hidden sequence, such as



This probability is

$$p(\mathbf{o}, \mathbf{s}) = a_{s_0} b_{s_0}(o_1) \prod_{i=2}^{T} a_{s_{i-1} s_i} b_{s_i}(o_i) \tag{2.12}$$

The first term inside the product in (2.12) is the probability that the ith state is s_i, given that the previous state was s_{i-1}; the second term is the probability of generating o_i from state s_i.

Calculating the probability of just the output sequence alone, however, requires at first glance summing (2.12) over the 2^T different possible hidden state sequences:

$$p(\mathbf{o}) = \sum_{\mathbf{s}} a_{s_0} b_{s_0}(o_1) \prod_{i=2}^{T} a_{s_{i-1} s_i} b_{s_i}(o_i) \tag{2.13}$$

Such an expensive operation is infeasible, but thankfully there is a more efficient way, using dynamic programming.

Figure 2.3 shows the trellis corresponding to all 2^T possible state sequences. At this point it is useful to introduce two helper variables:

$$\begin{aligned} \alpha_i^x &\stackrel{\text{def}}{=} p(o_1 o_2 \ldots o_{i-1}, s_i = x) \\ \beta_i^x &\stackrel{\text{def}}{=} p(o_i o_{i+1} \ldots o_T \mid s_i = x) \end{aligned}$$

In words, α_i^x is the probability that the HMM generates the first $i-1$ symbols from the

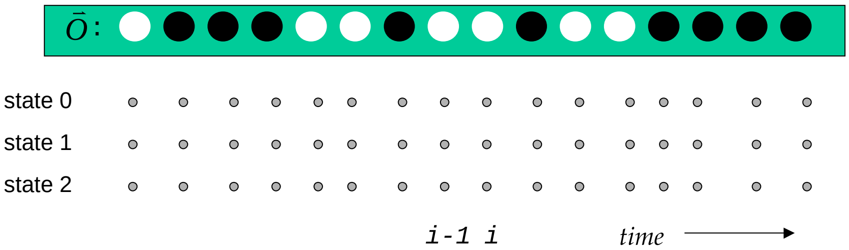


Figure 2.3: A trellis depicting, in compact form, all possible hidden state sequences in generating a sequence of T balls from the urns and mugs model.

output sequence $\mathbf{o}$, and after doing so winds up in state x. And β_i^x is the probability that, starting from state x, the HMM generates the suffix of $\mathbf{o}$ starting with o_i.

Notice that $p(\mathbf{o}) = \alpha_i^1\beta_i^1 + \alpha_i^2\beta_i^2$ for any $i \in \{1, 2, \ldots T\}$. In particular,

$$p(\mathbf{o}) = \alpha_T^1 + \alpha_T^2. \tag{2.14}$$

Notice also that α and β can be expressed recursively:

$$\alpha_i^1 = \alpha_{i-1}^1 b_1(o_i)a_{11} + \alpha_{i-1}^2 b_2(o_i)a_{21} \tag{2.15}$$

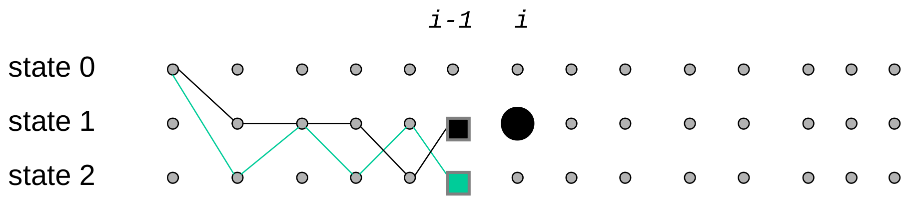
Equation (2.15) implies a linear-time calculation for α_1^T and α_2^T, which in turn implies (by inspecting (2.14)) a linear-time calculation for $p(\mathbf{o})$.

II. Decoding

We now pursue the second question: what's the most likely state sequence that a known HMM follows in generating $\mathbf{o}$? Of course, one could attempt to calculate $p(\mathbf{o}, \mathbf{s})$ for all 2^T possible paths $\mathbf{s}$, but there is a better way—known as the Viterbi algorithm [29]. This algorithm relies on the Markovian property of the state machine in the following way:

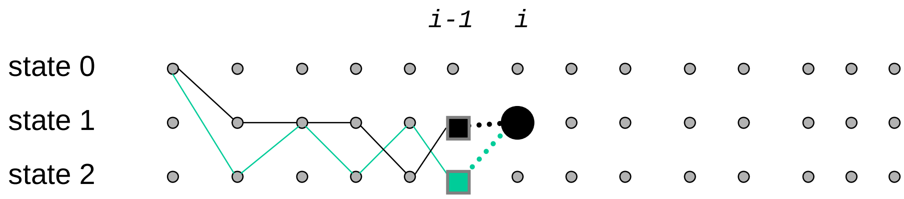
> *The most probable path ending in state x at time i contains, as its first $i-1$ entries, the most probable path ending at some state at time $i-1$.*

To begin, imagine that we know the most likely state sequence ending in state 1 at time $i-1$, and also the most likely state sequence ending in state 2 at time $i-1$:



Furthermore, denote by ϕ the joint probability $p(o_1 o_2 \ldots o_{i-1}, s_1 s_2 \ldots s_{i-2} 1)$ that the HMM follows the optimal path up to state 1 at time $i-1$, during which it generates the first $i-1$ symbols of $\mathbf{o}$. Similarly, let μ be the probability that the HMM follows the optimal path to state 2 at time $i-1$ and generates the appropriate prefix of $\mathbf{o}$ in so doing.

What is the best path ending at time i and state 1? By the underlying Markov property of the HMM, the only possible answers are represented by the dotted lines:



The lighter-colored candidate has probability μa_{21} and the darker-colored candidate has probability ϕa_{11}. Calculating the optimal path ending in state 1 at time i therefore requires a number of calculations which is only linear in the number of states, and (by applying the recursion T times) calculating the optimal path in generating all T symbols from $\mathbf{o}$ requires time proportional to $T|\ s\ |$. This recursive procedure is an implementation of the Viterbi algorithm [29].

III. Parameter estimation

We will now address the third question—that of parameter estimation. For simplicity, we'll focus attention on estimating the maximum-likelihood value for $b_1(\bullet)$: the probability of drawing a black ball from the first urn. (The other parameters are a_{x1} for $x \in \{0,1,2\}$.)

To begin, denote by $\gamma_t(1)$ the posterior probability that the HMM is in state 1 at time t while producing the output sequence $\mathbf{o}$. In terms of previously-defined quantities, $\gamma_t(1)$ is

$$\gamma_t(1) = p(s_t = 1 \mid \mathbf{o}) = \frac{\alpha_t^1 \beta_t^1}{p(\mathbf{o})}. \tag{2.16}$$

Given the observed output sequence $\{o_1 o_2 \ldots o_T\}$, the maximum-likelihood estimates of the numerator and denominator in the last term above may be written as

$$\begin{aligned} p(\text{output} = \bullet, \text{state} = 1) &= \frac{\sum_{t:o_t=\bullet} \gamma_t(1)}{T} \\ p(\text{state} = 1) &= \frac{\sum_t \gamma_t(1)}{T} \end{aligned}$$

Combining the above last three equalities, we have

$$\begin{aligned} b_1(\bullet) &= \frac{p(\text{output} = \bullet, \text{state} = 1)}{p(\text{state} = 1)} \\ &= \frac{\sum_{t:o_t=\bullet} \gamma_t(1)}{\sum_t \gamma_t(1)} \end{aligned} \tag{2.17}$$

Notice that in (2.16) and (2.17), $\gamma_t(1)$ is expressed in terms of $b_1(\bullet)$ and $b_1(\bullet)$ is expressed in terms of $\gamma_t(1)$. So we cannot calculate these quantities in closed form. But the mutually recursive definitions suggest an iterative algorithm, known as *Baum-Welch* or *forward-backward* estimation, summarized in Algorithm 4.

Algorithm 4: *Baum-Welch*

1. (*Initialization:*) `Pick a starting value` $b_1(\bullet) \in (0, 1)$
2. `Repeat until convergence:`

   (*E-step*) `Compute the expected number of times` $\bullet$ `is generated from state` 1 `in producing` $\mathbf{o}$`:`
   $E[\bullet \mid 1] = \sum_{t:\ o_t=\bullet} \gamma_t(1)$

   (*M-step*) `Calculate` $b_1(\bullet)$ `according to (2.17)`

As an instantiation of the EM algorithm, the Baum-Welch procedure inherits the attractive convergence guarantees of EM. The reader is referred to [5, 24, 69] for further details.

Chapter 3

Document ranking

This chapter applies statistical machine learning to the task of ordering documents by relevance to a query. The approach contains two main ingredients: first, novel probabilistic models governing the relation between queries and relevant documents, and second, algorithms for estimating optimal parameters for these models. The architecture and performance of a proof-of-concept system, called WEAVER*, is described. On a suite of datasets with very different characteristics,* WEAVER *exhibits promising performance, often with an efficiency approaching real-time. This chapter gives an information-theoretic motivation for these models, and shows how they generalize the recently proposed retrieval methods of language modeling and Hidden Markov Models.*

3.1 Problem definition

The goal in this chapter is to construct probabilistic models of language to address a core problem in information retrieval: ranking documents by relevance to a query.

The approach relies on the notion of *document distillation*. When a person formulates a query to an information retrieval system, what he is really doing (one could imagine) is distilling an information need into a succinct form. This distillation process begins with a document—containing the normal superfluence of textual fat and connective tissue such as prepositions, commas and so forth—and ends with a query, comprised of just those skeletal index terms characterizing the document. It may be that some of these index terms do not even appear in the document: one could easily imagine a newspaper article containing the words `Pontiff`, `mass` and `confession`, but never `Catholicism`, the single word which might best typify the document.

The strategy this chapter takes in assessing the relevance of a document to a query is to estimate the likelihood that a person would distill the document into the query. This "cart before the horse" view of retrieval is exactly that introduced in Section 2.1. This perspective is obviously not a faithful model of how a person formulates a query, yet it turns out to be a useful expedient. In formulating probabilistic models of the retrieval process, it appears easier to view the problem as a compression from document to query than an expansion from query to document. Moreover, the query-generative models proposed here can account, in a natural and principled way, for many of the features that are critical to modern high performance retrieval systems, including term weighting, query expansion, and length normalization. A prototype ranking system called WEAVER which employs these models demonstrates very promising empirical behavior, without sacrificing the nimble execution of more traditional approaches.

An ideal document retrieval system would contain at least enough linguistic sophistication to account for *synonymy* effects—to know, for instance, that `Pontiff` and `Pope` are related terms, and a document containing one may be related to a query containing the other. One could imagine equipping a relevancy ranking system with a database of such relations. Such a system would be more sophisticated, and hopefully more accurate, than one which adjudicated relevance solely on the basis of word overlap. Loosely speaking, this is the approach described here; this chapter contains algorithms for automatically constructing this database of word relations from a collection of documents.

In a sense, WEAVER has a pedigree in statistical translation, which concerns itself with how to mine large text databases to automatically discover such semantic relations. Brown *et al.* [13, 14] showed, for instance, how a computer can "learn" to associate French terms with their English translations, given only a collection of bilingual French/English sentences. The Candide system [6], an experimental project at IBM Research in the early 1990s, used the proceedings of the Canadian parliament, maintained in both English and French, to automatically learn to translate between these languages.

3.1.1 A conceptual model of retrieval

In formulating a query to a retrieval system, a user begins with an information need. This information need is represented as a fragment of an "ideal document"—a portion of the type of document that the user hopes to receive from the system. The user then translates or distills this ideal document fragment into a succinct query, selecting key terms and replacing some terms with related terms: replacing `pontiff` with `pope`, for instance.

Summarizing the model of query generation,

1. The user has an information need $\Im$.
2. From this need, he generates an ideal document fragment $\mathbf{d}_{\Im}$.
3. He selects a set of key terms from $\mathbf{d}_{\Im}$, and generates a query $\mathbf{q}$ from this set.

A reader might at this point be somewhat baffled at the notion that when fulfilling an information need, a person seeks a document most similar to a fragment $\mathbf{d}_{\Im}$ he has in his mind. After all, if the user *knew* what he was looking for, he wouldn't have the information need in the first place. What the user seeks is, in fact, exactly what he *doesn't* know.

To escape this apparent contradiction, we need to clearly define the notion of "user." For the purposes of this discussion, a user is someone who has a rough idea about what the desired document might contain. The user isn't the person who wants to learn the circumstances of Caesar's death, but rather the reference librarian who knows Caesar was assassinated by Longinus and Brutus on March 15, 44 B.C., and who would like a list of documents which roughly match the ideal document fragment `Caesar's assassination by Longinus and Brutus on March 15.`

One can view the imaginary process of query formulation as a corruption of the ideal document. In this setting, the task of a retrieval system is to find those documents most similar to $\mathbf{d}_{\Im}$. In other words, retrieval is the task of finding, among the documents comprising the collection, likely preimages of the user's query. Figure 3.1 depicts this model of retrieval in a block diagram.

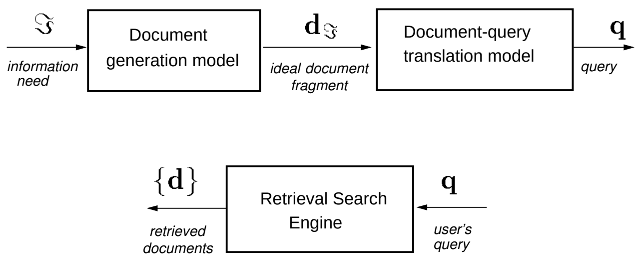


Figure 3.1: A conceptual view of query generation (above) and retrieval (below)

Figure 3.1 is drawn in a way that suggests an information-theoretic perspective. One can view the information need $\Im$ as a signal that gets corrupted as the user $\mathcal{U}$ distills it into a query $\mathbf{q}$. That is, the query-formulation process represents a noisy channel, corrupting the information need just as a telephone cable corrupts the data transmitted by a modem. Given $\mathbf{q}$ and a model of the channel—how an information need gets corrupted into a query—

the retrieval system's task is to identify those documents $\mathbf{d}$ that best satisfy the information need of the user.

More precisely, the retrieval system's task is to find the *a posteriori* most likely documents given the query; that is, those $\mathbf{d}$ for which $p(\mathbf{d} \mid \mathbf{q}, \mathcal{U})$ is highest. By Bayes' law,

$$p(\mathbf{d} \mid \mathbf{q}, \mathcal{U}) = \frac{p(\mathbf{q} \mid \mathbf{d}, \mathcal{U})\ p(\mathbf{d} \mid \mathcal{U})}{p(\mathbf{q} \mid \mathcal{U})}\,. \tag{3.1}$$

Since the denominator $p(\mathbf{q} \mid \mathcal{U})$ is fixed for a given query and user, one can ignore it for the purpose of ranking documents, and define the relevance $\rho_{\mathbf{q}}(\mathbf{d})$ of a document to a query as

$$\rho_{\mathbf{q}}(\mathbf{d}) = \underbrace{p(\mathbf{q} \mid \mathbf{d}, \mathcal{U})}_{\text{query-dependent}} \times \underbrace{p(\mathbf{d} \mid \mathcal{U})}_{\text{query-independent}}\,. \tag{3.2}$$

Equation (3.2) highlights the decomposition of relevance into two terms: first, a query-dependent term measuring the proximity of $\mathbf{d}$ to $\mathbf{q}$, and second, a query-independent or "prior" term, measuring the quality of the document according to the user's general preferences and information needs. Though this chapter assumes the prior term to be uniform over all documents, it's likely that in real-world retrieval systems the prior will be crucial for improved performance, and for adapting to a user's needs and interests. At the very least, the document prior can be used to discount "dubious" documents—those that are very short, or perhaps documents in a foreign language.[1]

Section 3.3 will contain a detailed formulation of two parametric models $p(\mathbf{q} \mid \mathbf{d})$, but as a preview, we outline here the four steps to model construction.

- *Data collection*: Start with a corpus of $(\mathbf{d}, \mathbf{q})$ pairs, where each pair consists of a query and a document relevant to the query. Acquiring a collection of $(\mathbf{d}, \mathbf{q})$ pairs from which to learn the document-to-query distillation model is a matter requiring some creativity. Insofar as the distillation model has a large number of parameters, robust estimation of these parameters requires a large document/query collection. This chapter will make use several different datasets. But in general it may be unrealistic to assume the existence of a large collection of query-document pairs, and so this chapter also introduces a strategy for overcoming a lack of labeled training data. Section 3.4 will describe a technique for synthesizing $(\mathbf{d}, \mathbf{q})$ pairs from a collection consisting solely of documents—unlabeled data, in other words—using statistical sampling.

[1] As a reminder, boldface letters refer to sequences of words—such as documents or queries—while italic letters denote single terms. So $p(q \mid \mathbf{d})$ is the probability of generating a *single* query word from an entire document $\mathbf{d}$, while $p(\mathbf{q} \mid \mathbf{d})$ is the probability that, in generating an entire query from the document $\mathbf{d}$, a user selected $\mathbf{q}$.

- *Model selection*: The core ingredient of the statistical distillation model introduced here is a square stochastic matrix of word-to-word "relatedness" probabilities. For a vocabulary of $20,000$ words, the matrix has size $400,000,000$, or 1.6GB if each parameter is a four-byte floating point value. Of course, sparse matrix techniques in practice reduce this size dramatically, and in fact Section 3.6 describes a lossy compression technique wherein the matrix requires space only *linear* in the size of the vocabulary.

- *Parameter estimation*: Given a collection of query, document pairs, use standard machine learning algorithms to estimate optimal parameter values for the parametric model $p(\mathbf{q} \mid \mathbf{d})$—a model of the likelihood that a sequence of words $\mathbf{q}$ is a distillation of (a translation of) a document $\mathbf{d}$.

- *Search*: Given a learned model $p(\cdot \mid \mathbf{d})$ and a new query $\mathbf{q}$, order documents by relevance to $\mathbf{q}$ by ranking them by decreasing $p(\mathbf{q} \mid \mathbf{d})$.

For an empirical evaluation of WEAVER, we report on experiments conducted on three different datasets: newswire articles drawn from the TREC corpus [84], a set of user transactions collected from the Lycos search engine, and a set of personal emails.

3.1.2 Quantifying "relevance"

The traditional IR view of "relevance to a query" is a property that a document may or may not enjoy. In other words, the relevance of a $\mathbf{d}$ to a given query $\mathbf{q}$ may be thought of as a binary random variable:

$$\rho_{\mathbf{q}}(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{d} \text{ is relevant to } \mathbf{q} \\ 0 & \text{otherwise} \end{cases}$$

The notation suggests a functional dependence just on $\mathbf{d}$ and $\mathbf{q}$, but in fact $\rho_{\mathbf{q}}(\mathbf{d})$ may depend on a number of other factors, including the person $\mathcal{U}$ using the system. After all, relevance is a subjective notion, and people may disagree about the "true" value of $\rho_{\mathbf{q}}(\mathbf{d})$.

Treating the relevance of a document to a query as a binary random variable has a long history in information retrieval. The earliest reference to a probabilistic approach to retrieval appears to be in Maron and Kuhns [58]. In their seminal work on relevance, Robertson and Sparck-Jones refer to the "Basic Question" of document ranking: "What is the probability that *this* document is relevant to *this* query?" [74]. They propose that the optimal document ranking algorithm relies on this probability of relevance. Their well-known "axiom" in IR is known as the *Probability Ranking Principle*:

> *If retrieved documents are ordered by decreasing probability of relevance on the data available, then the system's effectiveness is the best to be gotten for the data.*

The industry-standard *tfidf* score grew out from this philosophy; it was originally designed to distinguish between documents for which $\rho_{\mathbf{q}}(\mathbf{d}) = 1$ and those for which $\rho_{\mathbf{q}}(\mathbf{d}) = 0$.

This chapter also takes the Probability Ranking Principle as axiomatic, though it proposes a novel way to think about—and calculate—document relevance. WEAVER's retrieval strategy involves calculating a score, denoted by $p(\mathbf{d} \mid \mathbf{q})$, for estimating the relevance of document $\mathbf{d}$ to query $\mathbf{q}$. Unlike the *tfidf* score, $p(\mathbf{d} \mid \mathbf{q})$ is a probability distribution over documents, and therefore $\sum_{\mathbf{d}} p(\mathbf{d} \mid \mathbf{q}) = 1$. One can interpret $p(\mathbf{d} \mid \mathbf{q})$ as the probability that document $\mathbf{d}$ is the most relevant document in the collection for $\mathbf{q}$. One could argue that more is required of $p(\mathbf{d} \mid \mathbf{q})$ than of $\rho_{\mathbf{q}}(\mathbf{d})$: the former must impose a total ordering on documents, while the latter must only reveal a yes/no value for each document. In fact, one can reduce a $p(\mathbf{d} \mid \mathbf{q})$ ranking to a $\rho_{\mathbf{q}}(\mathbf{d})$ ranking, given a single real-valued relevance threshold; i.e., a cutoff value for $p(\mathbf{d} \mid \mathbf{q})$.

3.1.3 Chapter outline

The rest of this chapter will proceed as follows. Section 3.2 lays the groundwork by describing the language modeling approach to retrieval. Section 3.3 introduces two statistical models governing the distillation of documents to queries. Section 3.4 explains how one can estimate, via the EM algorithm, the parameters of such models automatically using just a collection of documents. Section 3.5 discusses the results of a set of experiments using three datasets: TREC newswire data, user transactions from a large commercial web search engine, and a collection of personal email correspondences, and also compares the proposed ranking algorithm to a more traditional vector-space technique: *tfidf* with Rocchio-based automatic query expansion.

The responsibility of a large-scale document retrieval system is to find those documents most relevant to a query in a spritely manner. One might think this need for speed precludes the use of "interesting" models for relevance scoring; after all, a retrieval system can't afford to get bogged down evaluating a complicated relevance metric for each document. However, Section 3.6 shows how, with a certain mix of preprocessing, time-space tradeoffs, and efficient data structures, WEAVER can can have its cake and eat it too: efficient retrieval with a non-trivial relevance function. Finally, Section 3.7 will suggest how the proposed statistical machine learning approach may be applicable to cross-language retrieval.

3.2 Previous work

The fields of information retrieval, bibliometrics, and language processing have been well populated over the years with research identifying itself as probabilistic in nature. As mentioned above, probabilistic approaches to IR date back at least forty years. Rather than attempting a survey of all this work, this section instead focus on two recently introduced probabilistic approaches most similar in spirit to the approach proposed in this chapter.

3.2.1 Statistical machine translation

The **Candide project** was a research project undertaken at IBM Thomas J. Watson Research Laboratories in the early 1990s to assess how far statistical machine learning techniques could go in constructing an automatic language translation system [6]. Starting from the proceedings of the Canadian parliament—conveniently transcribed in both English and French—the Candide system calculated parameter values for a statistical model of language-to-language translation. An electronically transcribed version of the Canadian parliament proceedings, known as Hansards, comprise several hundred million words and are an invaluable resource for machine learning and translation. Not only the retrieval system described in this chapter, but also the summarization system described in Chapter 4 owe an intellectual debt to Candide, both in the general sense of parameter estimation using text corpora, and, more specifically, in using the EM algorithm as a learning paradigm.

3.2.2 Language modeling

Statistical models of language are in common use in many language-related technologies, including automatic speech and handwriting recognition [42]. Ponte and Croft [67, 68] have recently proposed using language models for retrieval in the following way.

To each document in the collection, associate a probability distribution $l(\cdot \mid \mathbf{d})$ over words—in other words, a language model. Now imagine compressing the document $\mathbf{d}$ by selecting a size m for the smaller document, and then drawing m words at random from $\mathbf{d}$. The probability that this process will result in the new compressed document $\mathbf{c} = \{c_1, c_2 \ldots c_m\}$ is

$$p(\mathbf{c} \mid \mathbf{d}) = \phi(m) \prod_{i=1}^{m} l(c_i \mid \mathbf{d}) \tag{3.3}$$

Here $\phi(\cdot)$ is a distribution over lengths for the resulting compressed document.

The idea behind the language modeling approach to retrieval is to equate the relevance of a document to a query with the probability that the query would be generated by this process of compression applied to the document. Here again one can see, as in Section 3.1, a query-generative view of retrieval.

In the most straightforward implementation, the probability that a document will generate a word w is exactly the frequency of w in the document. But this isn't quite right, because it suggests that a document not containing w should never generate w. This amounts to saying that all documents not containing every word in the query are equally irrelevant—$l(\mathbf{q} \mid \mathbf{q}) = 0$—to the query.

One can avoid this situation by linearly interpolating or "smoothing" the frequency-based estimate with a model $l(\cdot \mid \mathcal{D})$ estimated using the *entire* collection $\mathcal{D}$ of documents, rather than just $\mathbf{d}$:

$$l_\alpha(w \mid \mathbf{d}) = \alpha l(w \mid \mathbf{d}) + (1-\alpha) l(w \mid \mathcal{D}) \tag{3.4}$$

The value of α can be estimated using standard machine learning techniques on a collection of data separate from that used to determine the $l(\cdot \mid \mathbf{d})$ and $l(\cdot \mid \mathcal{D})$ distributions.

With smoothing, (3.3) becomes

$$l(\mathbf{q} \mid \mathbf{d}) = \prod_{i=1}^{m} \alpha l(q_i \mid \mathbf{d}) + (1-\alpha) l(q_i \mid \mathcal{D}) \tag{3.5}$$

The predictive statistical models used in many language-related technologies are context-sensitive, meaning they assess the likelihood of a word appearing in text by inspecting the preceding words: `apple` is more likely when the previous word was `juicy` than when the previous word was, say, `hyperbolic`. However, the statistical model $l(\cdot \mid \mathbf{d})$ is *context-independent*, assuming naively that the author of a document generates the document by drawing words independently from a "bag of words." The issue of context-dependence comes up again in Chapter 4, but in that case the problem is addressed by modeling short-range dependencies between words in text.

3.2.3 Hidden Markov Models

This section discusses another recently-proposed query-generative model for retrieval which, although employing essentially the same scoring formula as the language modeling approach, arrives at this formula from a very different direction.

Miller *et al.* [61] propose using HMMs for retrieval in the following way. To each document $d \in \mathcal{D}$ in the collection, associate a distinct two-state HMM. The first of these states generates words w from the document $\mathbf{d}$ itself according to $l(w \mid \mathbf{d})$: if 10 percent of the words in $\mathbf{d}$ are w, then when the automaton is in this state it outputs the symbol w with probability $1/10$. The second state generates words according to $l(w \mid \mathcal{D})$, the frequency of w in the collection overall. The transitions within a two-state HMM are in general given by a two-by-two transition matrix, but by assuming the probability of moving to a state is independent of the current state, one can capture the transition probabilities using two values $\{a_1, a_2\}$, where a_i denotes the probability that the machine is in the ith state. Figure 3.2 illustrates such a two-state automaton. This machine is essentially identical to the "two urns and a mug" example of Section 2.3.1.

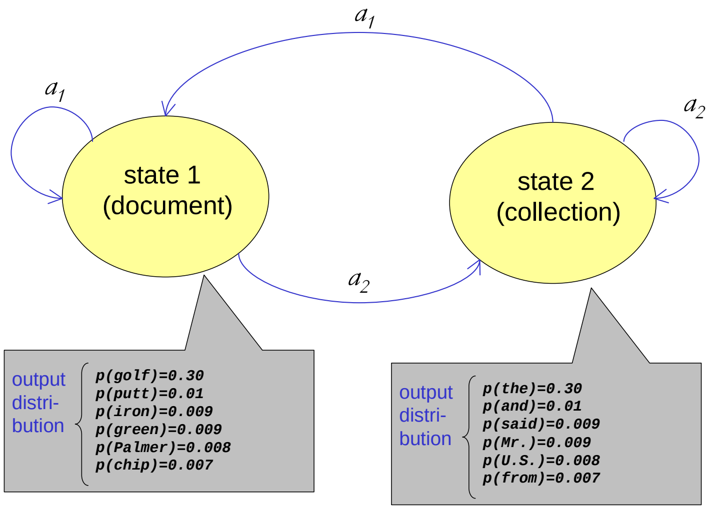


Figure 3.2: An idealized two-state Hidden Markov Model for document retrieval. To each document corresponds a distinct such automaton. The relevance of a document $\mathbf{d}$ to a query$\mathbf{q}$ is proportional to the likelihood that the probabilistic automaton for $\mathbf{d}$ will produce $\mathbf{q}$. Depicted is an imaginary automaton corresponding to a document about golf. While in the left (document) state the automaton outputs words according to their frequency $l(\cdot \mid \mathbf{d})$ in the document, and while in the right (collection) state, it outputs words according to their frequency $l(\cdot \mid \mathcal{D})$ in the collection.

As with the language modeling approach, document relevance for a query is equated with $p(\mathbf{q} \mid \mathbf{d})$: the probability, in this case, that the automaton for $\mathbf{d}$ will generate the query $\mathbf{q} = \{q_1, q_2, \ldots q_m\}$ ::

$$p(\mathbf{q} \mid \mathbf{d}) = \prod_{i=1}^{m} a_1 l(q_i \mid \mathbf{d}) + (1 - a_1) l(q_i \mid \mathcal{D}) \tag{3.6}$$

The HMM approach appears to be quite extensible: one could add more states, use a more sophisticated transition model, and account for word context by allowing states to output word sequences like bigrams (pairs of words) or trigrams. But in its two-state form with state-independent transitions, the HMM is in fact equivalent to the language modeling technique (with smoothing); this correspondence is obvious by inspection of (3.5) and (3.6).

* * *

The language modeling and Hidden Markov Model approaches together represent a novel and theoretically motivated "query-generative" approach to retrieval. Moreover, recent work has established the empirical performance of these techniques to be competitive or in some cases superior to standard *tfidf*-based retrieval. However, these approaches do not address (except as a post-processing step not integrated into the overall probabilistic model) the important issue of word-relatedness: accounting for the fact that `Caesar` and `Brutus` are related concepts. The word-relatedness problem has received much attention within the document retrieval community, and researchers have applied a variety of heuristic and statistical techniques—including pseudo-relevance feedback and local context analysis [28, 89].

This chapter will introduce a technique which generalizes both the LM and HMM approaches in such a way that the resulting model accounts for the word-relatedness phenomenon. Interpolating a document-model with a collection-wide distribution over words (as the LM and HMM approaches propose) ensures that no document assigns a zero probability to any word, but it does not acknowledge that a document containing `car` is likely to generate a query containing the word `automobile`. The next section will develop a general statistical framework for handling these issues.

3.3 Models of Document Distillation

Suppose that an information analyst is given a news article and asked to quickly generate a list of a few words to serve as a rough summary of the article's topic. As the analyst rapidly skims the story, he encounters a collection of words and phrases. Many of these he rejects as irrelevant, but his eyes rest on certain key terms as he decides how to render them in the summary. For example, when presented with an article about Pope John Paul II's visit to Cuba in 1998, the analyst decides that the words `Pontiff` and `Vatican` can simply be represented by the word `Pope`, and that `Cuba`, `Castro` and `island` can be collectively referred to as `Cuba`.

This section presents two statistical models of this query formation process, making specific independence assumptions to derive computationally and statistically efficient algorithms. While our simple query generation models are mathematically similar to those used for statistical translation of natural language [14], the duties of the models are qualitatively different in the two settings. Document-query distillation requires a compression of the document, while translation of natural language will tolerate little being thrown away.

3.3.1 Model 1: A mixture model

A "distillation model" refers to a conditional probability distribution $p(\mathbf{q} \mid \mathbf{d})$ over sequences of query words $\mathbf{q} = \{q_1, q_2 \ldots, q_m\}$, given a document $\mathbf{d} = \{d_1, d_2, \ldots, d_n\}$. The value $p(\mathbf{q} \mid \mathbf{d})$ is an estimate of the probability that, starting from the document $\mathbf{d}$, a person will distill $\mathbf{d}$ into $\mathbf{q}$.

Imagine that a person distills a document $\mathbf{d}$ into a query $\mathbf{q}$ as follows:

- Choose a length m for the query, according to a sample size model $\phi(m \mid \mathbf{d})$.
- For each position $j \in [1 \ldots m]$ in the query:
  - Choose a word $d_i \in \mathbf{d}$ in the document from which to generate the next query word.
  - Generate the next query word by "translating" d_i—i.e., by sampling from the distribution $\sigma(\cdot \mid d_i)$.

Following Brown *et al.* [13], an *alignment* between sequences of words is a graphical representation of which document words are responsible for the words in the query. One can also include in position zero of the document an artificial "null word," written `<null>`. The purpose of the null word is to generate spurious or content-free terms in the query, like the words in the phrase `Find all of the documents...`.

Using the alignment a, $p(\mathbf{q} \mid \mathbf{d})$ decomposes as

$$p(\mathbf{q} \mid \mathbf{d}) = \sum_a p(\mathbf{q}, a \mid \mathbf{d}) = \sum_a p(\mathbf{q} \mid a, \mathbf{d}) p(a \mid \mathbf{d}) \tag{3.7}$$

Imagining that each query word arises from exactly one document word, (3.7) becomes

$$p(\mathbf{q} \mid a, \mathbf{d}) = \prod_{i=1}^{m} \sigma(q_i \mid d_{a_i}) \tag{3.8}$$

Here d_{a_i} is the document word aligned with the ith query word, and $\sigma(q \mid d)$ is a parameter of the model—the probability that the document word d is paired with the query word

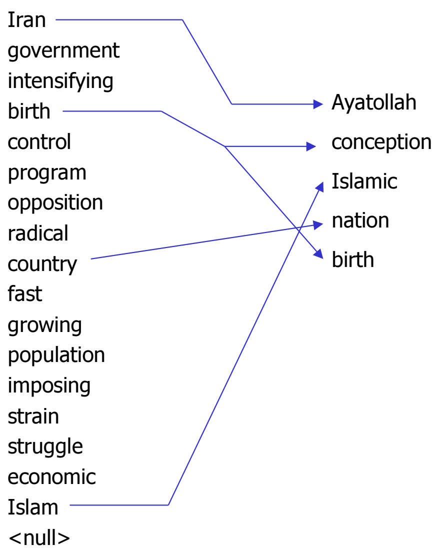


Figure 3.3: A word-to-word alignment of an imaginary document/query pair. The score of this single alignment, $p(\mathbf{q}, a \mid \mathbf{d})$, is a product of individual $\sigma(q_j \mid d_{a_j})$ word-to-word "relation" probabilities. Calculating the relevance of $\mathbf{d}$ to $\mathbf{q}$ involves summing the score of all alignments.

q in the alignment. Figure 3.3 depicts one of 5^{18} possible alignments of an imaginary document/query pair.

If $\mathbf{q}$ contains m words and $\mathbf{d}$ contains $n+1$ words (including the null word), there are $(n+1)^m$ alignments between $\mathbf{d}$ and $\mathbf{q}$. Assuming that all these alignments are *a priori* equally likely, one can write

$$p(\mathbf{q} \mid \mathbf{d}) = \frac{p(m \mid \mathbf{d})}{(n+1)^m} \sum_a \prod_{i=1}^{m} \sigma(f_i \mid e_{a_i}) \tag{3.9}$$

Given a collection of document/query pairs $\mathcal{C} = \{(\mathbf{q}_1, \mathbf{d}_1), (\mathbf{q}_2, \mathbf{d}_2), (\mathbf{q}_3, \mathbf{d}_3) \ldots\}$, the likelihood method suggests that one should adjust the parameters of (3.9) in such a way that the model assigns as high a probability as possible to $\mathcal{C}$. This maximization must be performed, of course, subject to the constraints $\sum_f \sigma(q \mid d) = 1$ for all words d. Using Lagrange multipliers,

$$\sigma(q \mid d) = \lambda^{-1} \sum_a p(\mathbf{q}, a \mid \mathbf{d}) \sum_{j=1}^{m} \delta(q, q_j)\delta(d, d_{a_j}), \tag{3.10}$$

where δ is the Kronecker delta function.

The parameter $\sigma(q \mid d)$ appears explicitly in the lefthand side of (3.10), and implicitly in the right. By repeatedly solving this equation for all pairs (q, d) (in other words, applying the EM algorithm), one eventually reaches a stationary point of the likelihood.

Equation (3.10) contains a sum over alignments, which is exponential and suggests that computing the parameters in this way is infeasible. In fact, this is not the case, since

$$\sum_a \prod_{i=1}^{m} \sigma(q_i \mid d_{a_i}) = \prod_{i=1}^{m} \sum_{j=0}^{n} \sigma(q_i \mid d_j) \tag{3.11}$$

This rearranging means that computing $\sum_a p(\mathbf{q}, a \mid \mathbf{d})$ requires only $\Theta(mn)$ work, rather than $\Theta(n^m)$.

We have already adopted the notation that $m \equiv |\,\mathbf{q}\,|$. Similarly, we will denote the length of the document by $n \equiv |\,\mathbf{d}\,|$. The probability $p(\mathbf{q} \mid \mathbf{d})$ is then the sum over all possible alignments, given by

$$p(\mathbf{q} \mid \mathbf{d}) = \frac{\phi(m \mid \mathbf{d})}{(n+1)^m} \sum_{a_1=0}^{n} \cdots \sum_{a_m=0}^{n} \prod_{j=1}^{m} \sigma(q_j \mid d_{a_j}). \tag{3.12}$$

(As a reminder, the range of a is from zero to n, rather than 1 to n, because the artificial null word lives in position zero of every document.)

A little algebraic manipulation shows that the probability of generating query $\mathbf{q}$ according to this model can be rewritten as

$$p(\mathbf{q} \mid \mathbf{d}) = \phi(m \mid \mathbf{d}) \prod_{j=1}^{m} \left(\frac{n}{n+1} p(q_j \mid \mathbf{d}) + \frac{1}{n+1} \sigma(w \mid \texttt{<null>}) \right) \tag{3.13}$$

where

$$p(q_j \mid \mathbf{d}) = \sum_w \sigma(q_j \mid w)\, l(w \mid \mathbf{d})\,,$$

with the document language model $l(w \mid \mathbf{d})$ given by relative counts. Thus, the query terms are generated using a mixture model—the document language model provides the mixing weights for the word-relation model, which has parameters $\sigma(q \mid w)$. An alternative view (and terminology) for this model is to describe it as a Hidden Markov Model, where the states correspond to the words in the vocabulary, and the transition probabilities between states are proportional to the word frequencies. The reader is invited to note the differences between this use of HMMs, depicted in Figure 3.4, and the two-state HMM of Figure 3.2.

The simplest version of Model 1, henceforth written as *Model 0*, is the one for which each word w can be mapped only to itself; that is, the word-relation probabilities are "diagonal":

$$\sigma(q \mid w) = \begin{cases} 1 & \text{if } q = w \\ 0 & \text{otherwise}\,. \end{cases}$$

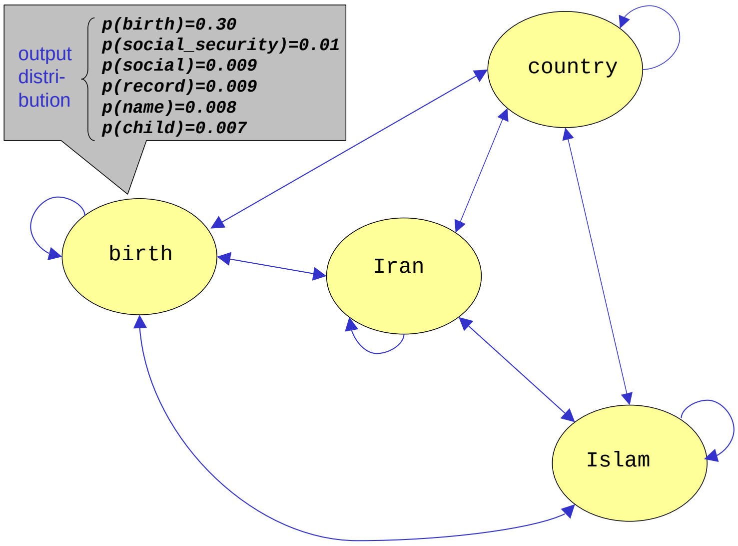


Figure 3.4: The document-to-query distillation process of Model 1 may be interpreted as a Hidden Markov Model: states represent words in the document, and the automaton moves to a state corresponding to word w according to the frequency of w in $\mathbf{d}$. The output distribution at state w is the EM-trained distribution over words: $\sigma(q \mid w)$ measures how closely word q is related to word w.

In this case, the query generation model is given by

$$p(q \mid \mathbf{d}) = \frac{n}{n+1} l(q \mid \mathbf{d}) + \frac{1}{n+1} \sigma(q \mid \texttt{<null>}) \, ,$$

a linear interpolation of the document language model and the background model associated with the null word.

3.3.2 Model 1′: A binomial model

This imaginary information analyst, when asked to generate a brief list of descriptive terms for a document, is unlikely to list multiple occurrences of the same word. To account for this assumption in terms of a statistical model, one can assume that a list of words is generated by making several independent translations of the document $\mathbf{d}$ into a single query term q, in the following manner. First, the analyst chooses a word w at random from the document. He chooses this word according to the document language model $l(w \mid \mathbf{d})$. Next, he translates w into the word or phrase q according to the word-relation model $\sigma(q \mid w)$. Thus, the probability of choosing q as a representative of the document $\mathbf{d}$ is

$$p(q \mid \mathbf{d}) = \sum_{w \in \mathbf{d}} l(w \mid \mathbf{d}) \, \sigma(q \mid w).$$

Assume that the analyst repeats this process n times, where n is chosen according to the sample size model $\phi(n \mid \mathbf{d})$, and that the resulting list of words is filtered to remove duplicates before it is presented as the summary, or query, $\mathbf{q} = q_1, q_2, \ldots q_m$.

Calculating the probability that a particular query $\mathbf{q}$ is generated in this way requires a sum over all sample sizes n, and consider that each of the terms q_i may have been generated multiple times. Thus, the process described above assigns to $\mathbf{q}$ a total probability

$$p(\mathbf{q} \mid \mathbf{d}) = \sum_n \phi(n \mid \mathbf{d}) \sum_{n_1 > 0} \cdots \sum_{n_m > 0} \binom{n}{n_1 \cdots n_m} \prod_{i=1}^{m} p(q_i \mid \mathbf{d})^{n_i}$$

This expression can be calculated efficiently using simple combinatorial identities and dynamic programming techniques. But instead of pursuing this path, assume that the number of samples n is chosen according to a Poisson distribution with mean $\lambda(\mathbf{d})$:

$$\phi(n \mid \mathbf{d}) = e^{-\lambda(\mathbf{d})} \frac{\lambda(\mathbf{d})^n}{n!}.$$

Making this assumption means that $p(\mathbf{q} \mid \mathbf{d})$ can be rewritten as

$$p(\mathbf{q} \mid \mathbf{d}) = e^{-\lambda(\mathbf{d})} \sum_n \lambda(d)^n \sum_{n_1 > 0} \cdots \sum_{n_m > 0} \frac{1}{n_1! n_2! \ldots n_m!} \prod_{i=1}^{m} p(q_i \mid \mathbf{d})^{n_i}$$

Note that since $n = \sum_{i=1}^{m} n_i$,

$$\lambda(\mathbf{d})^n = \lambda(\mathbf{d})^{n_1 + n_2 + \ldots n_m}$$

Using this fact, distribute the $\lambda(\mathbf{d})^n$ over the inner sums and do away with the sum over n:

$$p(\mathbf{q} \mid \mathbf{d}) = e^{-\lambda(\mathbf{d})} \sum_{n_1 > 0} \cdots \sum_{n_m > 0} \frac{1}{n_1! n_2! \ldots n_m!} \prod_{i=1}^{m} p(q_i \mid \mathbf{d})^{n_i} \lambda(d)^{n_i}$$

Rewriting the sum over n_1,

$$p(\mathbf{q} \mid \mathbf{d}) = e^{-\lambda(\mathbf{d})} \sum_{n_1} \frac{1}{n_1!} p(q_1 \mid \mathbf{d})^{n_1} \sum_{n_2 > 0} \cdots \sum_{n_m > 0} \left(\frac{1}{n_2! \ldots n_m!} \right) \prod_{i=2}^{m} p(q_i \mid \mathbf{d})^{n_i} \lambda(d)^{n_i}$$

Similarly, one can expand the rest of the n_i, yielding

$$p(\mathbf{q} \mid \mathbf{d}) = e^{-\lambda(\mathbf{d})} \prod_{i=1}^{m} \sum_{n_i} \frac{1}{n_i!} p(q_i \mid \mathbf{d})^{n_i} \lambda(d)^{n_i}$$

Finally, apply the Taylor series expansion of e^x to get

$$p(\mathbf{q} \mid \mathbf{d}) = e^{-\lambda(\mathbf{d})} \prod_{i=1}^{m} \left(e^{\lambda(d) p(q_i \mid \mathbf{d})} - 1 \right), \tag{3.14}$$

This formula shows that the probability of the query is given as a product of terms. Yet the query term translations are *not* independent, due to the process of filtering out the generated list to remove duplicates. The model expressed in equation (3.14) will henceforth be denoted as *Model 1′*.

Model 1′ has an interpretation in terms of binomial random variables. Suppose that a word w does *not* belong to the query with probability $\beta_w = e^{-\lambda(\mathbf{d})p(w \mid \mathbf{d})}$. Then Model 1′ amounts to flipping independent β_w-biased coins to determine which set of words comprise the query [36]. That is, the probability $p(\mathbf{q} \mid \mathbf{d})$ of equation (3.14) can be expressed as

$$p(\mathbf{q} \mid \mathbf{d}) = \prod_{w \in \mathbf{q}} (1 - \beta_w) \prod_{w \notin \mathbf{q}} \beta_w \,.$$

This model was inspired by another IBM statistical translation model, one that was designed for modeling a bilingual dictionary [15].

Model 1′ also has an interpretation in the degenerate case of diagonal word-relation probabilities. To see this, let us make a further simplification by fixing the average number of samples to be a constant λ independent of the document $\mathbf{d}$, and suppose that the expected number of times a query word is drawn is less than one, so that $\max_i \lambda l(q_i \mid \mathbf{d}) < 1$. Then to first order, the probability assigned to the query according to Model 1′ is a constant times the product of the language model probabilities:

$$p(\mathbf{q} = q_1, \ldots, q_m \mid \mathbf{d}) \approx e^{-\lambda} \lambda^m \prod_{i=1}^{m} l(q_i \mid \mathbf{d}) \,. \tag{3.15}$$

Since the mean λ is fixed for all documents, the document that maximizes the righthand side of the above expression is that which maximizes $\prod_{i=1}^{m} l(q_i \mid \mathbf{d})$. And this should look familiar: it's proportional to the language modeling score given in 3.3.

3.4 Learning to rank by relevance

The key ingredient in the models introduced in the previous section is the collection of word-relation probabilities $\sigma(q \mid w)$. A natural question to ask at this point is how to obtain these probabilities. One strategy is to learn these values automatically from a collection of data, using the likelihood criterion. Ideal would be a collection of query/document pairs to learn from, obtained by human relevance judgments; in other words, a collection of pairs $(\mathbf{q}, \mathbf{d})$ where in each pair the document $\mathbf{d}$ is known to be relevant to the query $\mathbf{q}$. We report in Section 3.5 on the use of several different datasets for this purpose. But in practice it may occur that no suitably large collection of query/document pairs exists from which to robustly estimate the model parameters, and so here we describe a method for learning values from just a collection of documents, which is considerably easier to acquire.

3.4.1 Synthetic training data

From a collection of documents, one can tease out the semantic relationships among words by generating *synthetic queries* for a large collection of documents and estimating the word-relation probabilities from this synthetic data.

At a high level, the idea is to take a document and synthesize a query to which the document would be relevant. There are a number of candidate methods for synthesizing a query from a document. One could sample words uniformly at random from the document, but this scheme would generate queries containing a disproportionate number of common words like `the, of, and, but`. Preferable would be a sampling algorithm biased in favor of words which distinguish the document from other documents.

To explain the rationale for the scheme applied here, we return to the fictitious information analyst, and recall that when presented with a document $\mathbf{d}$, he will tend to select terms that are suggestive of the content of the document. Suppose now that he himself selects an arbitrary document $\mathbf{d}$ from a database $\mathcal{D}$, and asks us to guess, based only upon his summary $\mathbf{q}$, which document he chose. The amount by which one is able to do better, on average, than randomly guessing a document from $\mathcal{D}$ is the *mutual information* $I(D;Q) = H(D) - H(D \mid Q)$ between the random variables representing his choice of document D and query Q [42]. Here $H(D)$ is the entropy in the analyst's choice of document, and $H(D \mid Q)$ is the conditional entropy of the document given the query. If he is playing this game cooperatively, he will generate queries for which this mutual information is large.

With this game in mind, one can take a collection of documents $\mathcal{D}$ and, for each document $\mathbf{d} \in \mathcal{D}$, compute the mutual information statistic [42] for each of its words according to

$$I(w, \mathbf{d}) = p(w, \mathbf{d}) \log \frac{p(w \mid \mathbf{d})}{p(w \mid \mathcal{D})}.$$

Here $p(w \mid \mathbf{d})$ is the probability of the word in the document, and $p(w \mid \mathcal{D})$ is the probability of the word in the collection at large. By scaling these $I(w, \mathbf{d})$ values appropriately, one can construct an artificial cumulative distribution function $\tilde{I}$ over words in each document. Drawing $m \sim \phi(\cdot \mid \mathbf{d})$ random samples from the document according to this distribution results in a query $\mathbf{q} = q_1, \ldots, q_m$. Several such queries were generated for each document.

In some sense, query generation is just a version of query reformulation, where the original query is empty. Taking this view brings into scope the large body of work in the IR community on query reformulation. The popular Rocchio relevance feedback technique, for instance, is a method for refining a query by examining the set of documents known to be relevant—and also a set known *not* to be relevant—to that query [75]. We will revisit query expansion techniques later in this chapter.

q	$\sigma(q \mid w)$
ibm	0.674
computer	0.042
machine	0.022
analyst	0.012
software	0.011
workstation	0.007
stock	0.006
system	0.006
business	0.005
market	0.005

$w =$ ibm

q	$\sigma(q \mid w)$
defend	0.676
trial	0.015
case	0.012
court	0.011
charge	0.011
judge	0.010
attorney	0.009
convict	0.007
prosecutor	0.006
accuse	0.006

$w =$ defend

q	$\sigma(q \mid w)$
whittaker	0.535
climber	0.048
everest	0.032
climb	0.023
expedition	0.018
garbage	0.015
chinese	0.015
peace	0.015
cooper	0.013
1963	0.012

$w =$ whittaker

q	$\sigma(q \mid w)$
solzhenitsyn	0.319
citizenship	0.049
exile	0.044
archipelago	0.030
alexander	0.025
soviet	0.023
union	0.018
komsomolskaya	0.017
treason	0.015
vishnevskaya	0.015

$w =$ solzhenitsyn

q	$\sigma(q \mid w)$
carcinogen	0.667
cancer	0.032
scientific	0.024
science	0.014
environment	0.013
chemical	0.012
exposure	0.012
pesticide	0.010
agent	0.009
protect	0.008

$w =$ carcinogen

q	$\sigma(q \mid w)$
unearth	0.816
bury	0.033
dig	0.018
remains	0.016
find	0.012
body	0.010
bone	0.007
death	0.004
site	0.003
expert	0.003

$w =$ unearth

q	$\sigma(q \mid w)$
pontiff	0.502
pope	0.169
paul	0.065
john	0.035
vatican	0.033
ii	0.028
visit	0.017
papal	0.010
church	0.005
flight	0.004

$w =$ pontiff

q	$\sigma(q \mid w)$
everest	0.439
climb	0.057
climber	0.045
whittaker	0.039
expedition	0.036
float	0.024
mountain	0.024
summit	0.021
highest	0.018
reach	0.015

$w =$ everest

q	$\sigma(q \mid w)$
wildlife	0.705
fish	0.038
acre	0.012
species	0.010
forest	0.010
environment	0.009
habitat	0.008
endangered	0.007
protected	0.007
bird	0.007

$w =$ wildlife

Figure 3.5: Sample EM-trained word-relation probabilities learned from a corpus of newswire articles collected from the NIST-sponsored TREC project [84].

3.4.2 EM training

The resulting corpus $\{(\mathbf{d}, \mathbf{q})\}$ of documents and synthetic queries was used to fit the probabilities of Models 1 and 1′ with the EM algorithm [24], run for only three iterations to avoid overfitting. A sample of the resulting word-relation probabilities, when trained on the *Associated Press* (AP) portion of the TREC volume 3 corpus, is shown in Figure 3.5. In this figure, a document word is shown together with the ten most probable query words that it will map to according to the model.

For these experiments, a 132,625-word vocabulary was used. In principle, the word-relatedness matrix corresponding to this vocabulary has 17.5 billion parameters. But enforcing that $\sigma(\mathbf{q} \mid \mathbf{d}) = 0$ for all pairs of word $(\mathbf{q}, \mathbf{d})$ which did not co-occur in a query/document pair in the training corpus reduced the number of free parameters to 47,065,200. Maximum likelihood values estimates for these parameters were calculated from a corpus obtained by generating five synthetic mutual information queries for each of the 78,325 documents in the collection.

Specifically, the data-generation process was as follows:

1. Do for each document $\mathbf{d} \in \mathcal{D}$:
   - Do for $x = 1$ to 5:
     - Select a length m for this query according to $\phi(\cdot \mid \mathbf{d})$
     - Do for $i = 1$ to m:
       - Select the next query word by sampling the scaled distribution: $q_i \sim \tilde{I}$
     - Record this $(\mathbf{d}, \mathbf{q})$ pair

For statistical models of this form, *smoothing* or interpolating the parameters away from their maximum likelihood estimates is important. One can use a linear interpolation of the background unigram model and the EM-trained word-relation model:

$$\begin{aligned} p_\alpha(q \mid \mathbf{d}) &= \alpha\, p(q \mid \mathcal{D}) + (1-\alpha)\, p(q \mid \mathbf{d}) \\ &= \alpha\, p(q \mid \mathcal{D}) + (1-\alpha) \sum_{w \,\in\, \mathbf{d}} l(w \mid \mathbf{d})\, \sigma(q \mid w)\,. \end{aligned} \tag{3.16}$$

The weight was empirically set to $\alpha = 0.05$ on heldout data. The models for the baseline language modeling approach, Model 0, were also smoothed using linear interpolation:

$$l_\gamma(w \mid \mathbf{d}) = \gamma\, p(w \mid \mathcal{D}) + (1-\gamma)\, l(w \mid \mathbf{d})\,.$$

This interpolation weight was fixed at $\gamma = 0.1$. The Poisson parameter for the sample size distribution was fixed at $\lambda = 15$, independent of the document. No adjustment of any parameters, other than those determined by unsupervised EM training of the word-relation probabilities, was carried out on the experiments described below.

Algorithm 5 is a method for estimating, given a query $\mathbf{q}$ and a large collection of documents $\mathbf{d}$, the relevance $\rho_{\mathbf{q}}(\mathbf{d})$ of each document to the query. The procedure—completely impractical for large-scale IR datasets—is to visit each document $\mathbf{d}$ in the collection and compute $p(\mathbf{q} \mid \mathbf{d})$ for each, according to (3.16). Section 3.6 takes up the matter of ranking documents efficiently, using an inverted index and an approximation to $p(q \mid \mathbf{d})$.

Algorithm 5: *"NaiveRank" document ranking*

Input: `Query` $\mathbf{q} = \{q_1, q_2, \ldots q_m\}$;
`Collection of documents` $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots \mathbf{d}_N\}$;
`Word-relation probability` $\sigma(q \mid w)$ `for all word pairs` q, w

Output: `Relevance score` $\rho_{\mathbf{q}}(\mathbf{d})$ `for each document` $\mathbf{d}$

1. `Do for each document` $d \in \mathcal{D}$ `in the collection`
2. `Set` $\rho_{\mathbf{q}}(\mathbf{d}) \leftarrow 1$
3. `Do for each query word` $q \in \mathbf{q}$:
4. `Calculate` $p_\alpha(q \mid \mathbf{d})$ `according to (3.16)`
5. `Set` $\rho_{\mathbf{q}}(\mathbf{d}) = \rho_{\mathbf{q}}(\mathbf{d}) \times p_\alpha(q \mid \mathbf{d})$

3.5 Experiments

This section describes the results of experiments conducted using WEAVER with a heterogeneous set of queries and documents. The document datasets employed here include two corpora of newswire articles, a set of transactions with a large commercial web search engine, and a set of personal emails. We also devote some attention to a comparison against traditional vector space methods.

Some of the questions these experiments address will include:

- How does the length of the query affect the behavior of WEAVER?
- How does the size of the document corpus affect the behavior of WEAVER?

- How does the type of document—newswire articles, web pages, or email—affect the behavior of WEAVER?
- What is the difference in practice between Model 0, Model 1 and *tfidf*?
- What is the difference in practice between Model 0 and the traditional language model ranking; in other words, how good is the approximation in (3.15)?
- In practice, what is an appropriate number of iterations for EM training for Model 1?

3.5.1 TREC data

The experiments here examine the behavior of the various candidate ranking algorithms on long queries, drawn from the concept fields of TREC topics 51-100, and short queries, drawn from the title fields of these same topics. Typically, the concept field of a TREC topic comprises 20 or more keywords, while the title field is much more succinct—usually not more than four words. The rather exhaustive concept field queries are perhaps not atypical of a query submitted by a librarian or expert information scientist, though certainly longer than "real-world" queries submitted, for instance, by users of commercial search engines. The latter are more similar to the TREC title fields. For illustration, a full TREC topic appears in Figure 3.6.

The experiments in this section use two main document collections: a set of 78,325 Associated Press (AP) articles and another set of 90,250 *San Jose Mercury News* (SJMN) articles. A separate set of experiments was conducted on a much smaller collection of 2,866 broadcast news transcripts from the *Spoken Document Retrieval* (SDR) track of the 1998 TREC evaluation. All of the data were preprocessed by converting to upper case, stemming using the Porter stemmer [70], and filtering with a list of 571 stopwords from the SMART system.

Precision-recall curves for the AP and SJMN data, generated from the output of the TREC evaluation software, appear in a series of figures and tables starting with Figure 3.7. The baseline curves in these plots show the performance of the *tfidf* measure using a commonly-used *tf* score [67]. They also show the result of using Model 0 to score the documents, suppressing the word-relation component of Model 1.

The first set of plots, depicted in Figure 3.7, illustrate the relative precision-recall performance of Models 1, $1'$ and Model 0, using the AP and SJMN collections. Figure 3.7 contains the exact values corresponding to the AP plot.

Domain: International Economics

Topic: Airbus Subsidies

Description: Document will discuss government assistance to Airbus Industrie, or mention a trade dispute between Airbus and a U.S. aircraft producer over the issue of subsidies.

Summary: Document will discuss government assistance to Airbus Industrie, or mention a trade dispute between Airbus and a U.S. aircraft producer over the issue of subsidies.

Narrative: A relevant document will cite or discuss assistance to Airbus Industrie by the French, German, British or Spanish government(s), or will discuss a trade dispute between Airbus or the European governments and a U.S. aircraft producer, most likely Boeing Co. or McDonnell Douglas Corp., or the U.S. government, over federal subsidies to Airbus.

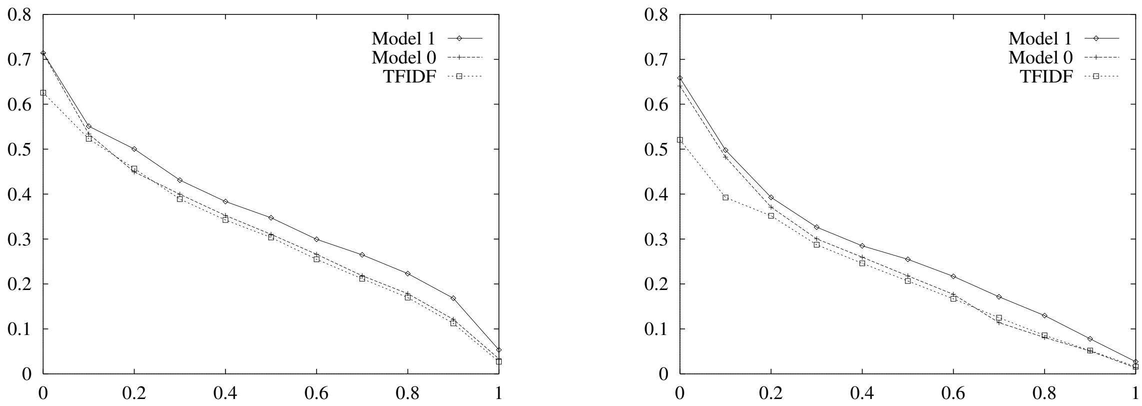
Concept(s):

1. Airbus Industrie
2. European aircraft consortium, Messerschmitt-Boelkow-Blohm GmbH, British Aerospace PLC, Aerospatiale, Construcciones Aeronauticas S.A.
3. federal subsidies, government assistance, aid, loan, financing
4. trade dispute, trade controversy, trade tension
5. General Agreement on Tariffs and Trade (GATT) aircraft code
6. Trade Policy Review Group (TPRG)
7. complaint, objection
8. retaliation, anti-dumping duty petition, countervailing duty petition, sanctions

Figure 3.6: An example topic (51) from the TREC collection. Document ranking systems often behave quite differently on short and long queries, and so this chapter includes evaluation results on both types, using the shorter title and more explicit concept fields of TREC topics 51-100.

	tfidf	Model 1	%Δ
Relevant:	5845	5845	—
Rel.ret.:	5845	5845	—
Precision:			
at 0.00	0.6257	0.7125	+13.9
at 0.10	0.5231	0.5916	+13.1
at 0.20	0.4569	0.5217	+14.2
at 0.30	0.3890	0.4554	+17.1
at 0.40	0.3425	0.4119	+20.3
at 0.50	0.3035	0.3636	+19.8
at 0.60	0.2549	0.3148	+23.5
at 0.70	0.2117	0.2698	+27.4
at 0.80	0.1698	0.2221	+30.8
at 0.90	0.1123	0.1580	+40.7
at 1.00	0.0271	0.0462	+70.5
Avg.:	0.2993	0.3575	+19.4
Precision at:			
5 docs:	0.4809	0.5574	+15.9
10 docs:	0.4702	0.5170	+10.0
15 docs:	0.4326	0.5135	+18.7
20 docs:	0.4213	0.4851	+15.1
30 docs:	0.3894	0.4539	+16.6
100 docs:	0.2960	0.3419	+15.5
200 docs:	0.2350	0.2653	+12.9
500 docs:	0.1466	0.1610	+9.8
1000 docs:	0.0899	0.0980	+9.0
R-Precision:	0.3254	0.3578	+10.0

Table 3.1: Performance of *tfidf* versus Model 1 for queries constructed from the concept fields. These numbers correspond to left plot in Figure 3.7.



AP-Concept **SJMN-Concept**

Figure 3.7: Comparing the performance of Models 1 and $1'$ to the baseline *tfidf* and Model 0 performance on AP data (left) and San Jose Mercury News document collections (right) when ranking documents for queries formulated from the TREC "concept" fields for topic 51-100.

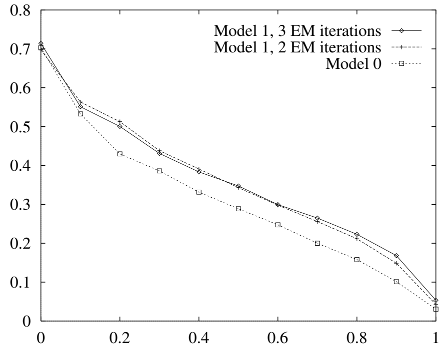


Figure 3.8: The discrepancy between two and three EM iterations of training for Model $1'$.

As with all statistical parameter estimation, overfitting during EM training is a concern. Figure 3.9 shows the performance of Model $1'$ on the AP data when the probabilities are trained for two and three iterations. The rather minor performance difference between these two curves suggests that only a small number of iterations are required for convergence for these models.

To study the effects of query length on WEAVER's performance, we also scored the documents for the title fields of topics 51–100, where the average query length is only 2.8 words. Comparing with Figure 3.7 (the corresponding performance for long queries using the same document collection) reveals that all candidate ranking algorithms deteriorate in

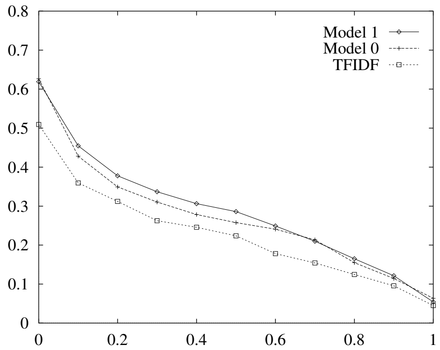


Figure 3.9: Comparing *tfidf*, Model 0, and Model 1 on short, title-field queries with Associated Press documents.

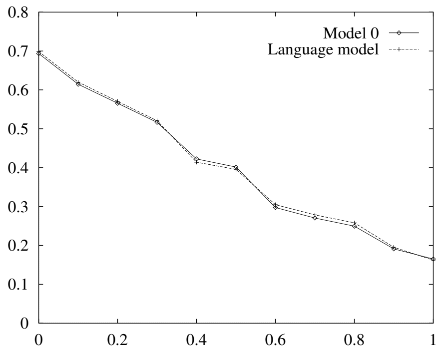


Figure 3.10: Comparing Model 0 to the "traditional" language model score using the product $\prod_{i=1}^{m} l(q_i \mid \mathbf{d})$.

performance. This is, of course, to be expected. What is notable is the substantial relative improvement of Model 1 over the *tfidf* baseline: 30.2% in average precision and 17.8% in R-precision on these short queries. The marginal improvement of Model 1 over Model 0 is smaller herme—6.3% in average precision and 4.9% in R-precision.

Figure 3.10 compares Model 0 with the traditional language model scoring score $\prod_i l(q_i \mid \mathbf{d})$, as in Ponte and Croft's method. The curves are essentially indistinguishable, suggesting that the approximation in equation (3.15) is good.

The TREC evaluation methodology is popular within the information sciences community for a number of reasons. First, the TREC datasets come with human-assigned relevance judgments, a rare and valuable commodity in the IR community. Second, the documents—mostly newswire articles—are rather uniform in size and style. But perhaps most important is that, for better or worse, the TREC evaluation method has become a widely recognized benchmark for document ranking systems.

Of course, a ranking algorithm which exhibits promising behavior on the TREC dataset may not perform as well in more rugged terrain. The next two sections describe the results of experiments conducted on datasets with markedly different characteristics from TREC data:

- A collection of user queries to the Lycos search engine along with the web page selected by the user from among those suggested by Lycos;
- A set of email subject lines (corresponding to queries) and bodies (corresponding to documents).

3.5.2 Web data

For the purposes of this section, a "clickthrough record" refers to a query submitted to the Lycos search engine, and the URL selected by the submitter from among the choices presented by Lycos. Table 3.2 lists a small extract of clickthrough records from a Lycos log in the early months of 2000.

From a large collection of clickthrough records similar to that in Table 3.2, we fetched the contents of each URL. Doing so gives a set of (query, web page) records: each web page is relevant to its associated query. More precisely, a Lycos user *suspected* the document to be relevant, based on what information the user could glean from the URL of the document and Lycos's capsule summary of the document. The idea, then, is to view the clickthrough data as a collection of human relevance judgments.

The experiments reported in this section relied on a simple filter to detect and remove those records containing objectionable content such as pornography and hate speech. This eliminated about 20% of the records gathered, leaving 630, 812 records. Other than mapping the query characters to lowercase, no processing of the queries submitted to Lycos was performed. A space-delimited sequence of characters represents a single term.

Avoiding preprocessing of the search engine's logfile underscores the "pushbutton" nature of WEAVER's retrieval algorithm. However, the data cry out for at least a minimal amount of preprocessing: the query in the second entry of Table 3.2, for instance, contains a comma, which distinguishes it from the semantically identical `missoula mt`. And the ninth query in the table contains an easily-detectable misspelling, which distinguishes

Query	**Selected URL**
`felony`	`jud13.flcourts.org/felony.html`
`missoula, mt`	`missoula.bigsky.net/score/`
`feeding infants solid foods`	`members.tripod.com/drlee90/solid.html`
`colorado lotto results`	`www.co-lotto.com/`
`northern blot`	`www.invitrogen.com/expressions/1196-3.html`
`wildflowers`	`www.life.ca/nl/43/flowers.html`
`ocean whales`	`playmaui.com/ocnraftn.html`
`ralph lauren polo`	`www.shopbiltmore.com/dir/stores/polo.htm`
`bulldog hompage`	`www.adognet.com/breeds/2abulm01.html`
`lyrics`	`www.geocities.com/timessquare/cauldron/8071`
`churches in atlanta`	`acme-atlanta.com/religion/christn.html`
`retail employment`	`www.crabtree-evelyn.com/employ/retail.html`
`illinois mortgage brokers`	`www.birdview.com/ypages2/c3.htm`
`stock exchange of singapore`	`www.ses.com.sg`
`front office software`	`www.saleslogic.com/saleslogix.phtml`
`free 3d home architect`	`www.adfoto.com/ads1/homeplans.shtml`
`country inns sale`	`innmarketing.com/form.html`
`free desktop wallpaper`	`www.snap-shot.com/photos/fireworks/`
`automotive marketing research`	`www.barndoors.com/rcmresources.htm`
`router basics`	`www.wheretobuy.com/prdct/706/55.html`

Table 3.2: A sample of Lycos clickthrough records—user query and the selected URL—during a several-second window in February, 2000.

it from `bulldog homepage`. But since Model 1 can ferret out correlations among terms, correctly and misspelled ones, preprocessing becomes rather less important.

The web pages were subject to rather more severe preprocessing. A publicly-available tool (`lynx`) siphoned off markup tags, images, and all other components of a web page besides plain text. A separate filter then removed punctuation, lowercased the text, and truncated all retrieved web pages to 2048 bytes, for efficiency.

The data were split into three parts by assigning each record, randomly, to one of the following disjoint sets:

- *Training*: $624,491$ (query, document) pairs
- *Heldout*: 1000 pairs
- *Evaluation*: 5321 pairs

Ranking web pages involved a linear interpolation of three models:

$$p(q \mid \mathbf{d}) = \alpha\, \sigma(q \mid \mathbf{d}) + \beta l(q \mid \mathbf{d}) + \gamma l(q \mid \mathcal{D}) \tag{3.17}$$

Estimating α, β, and γ worked as follows. First, we fixed $\alpha = 0$ and determined the optimal ratio between β and γ by line search in one dimension on the heldout data. Surprisingly, the optimal ratio turned out to be $\beta = 0.99$, $\gamma = 0.01$. One intuition for this result is as follows. Queries to commercial web search engines like Lycos tend to be very short; the average query length in the Lycos training data, for instance, was 2.6 words. The model $l(q \mid \mathcal{D})$ is especially useful for longer queries, to help a relevant document overcome a "missing" query word. But with short (and especially single-word) queries, a document is less likely to be relevant if it doesn't contain all the terms in the query.

After fixing the $\beta : \gamma$ ratio, we calculated α by applying a line search for the best $\alpha : (\beta + \gamma)$ ratio, which turned out to be $6 : 4$. The final ratio was therefore $\alpha = 0.4, \beta = 0.594, \gamma = 0.006$.

Using standard precision/recall metrics to evaluate WEAVER's performance on this dataset makes little sense, because here there exists only a single relevant document for each query. Instead of using precision/recall, we concentrate on the rank of the single known relevant document within the relevancy ranking produced by the system. Putting these ranks together gives a vector of ranks, where the i'th entry is the rank, according to WEAVER, of the known relevant document for query i.

There are a number of different reasonable metrics to use in evaluating a list of ranks. The median value in the vector is one reasonable metric; another is the inverse harmonic mean rank. From a set of rankings $\{r_1, r_2, \ldots r_N\}$, one can measure the inverse harmonic mean rank as follows:

$$M \stackrel{\text{def}}{=} \frac{N}{\sum_{i=1}^{N} \frac{1}{r_i}}$$

A lower number indicates better performance; $M = 1$, which is optimal, means that the algorithm consistently assigns the first rank to the correct answer.

Table 3.3 contains the results from Model 0 and Model 1 on the Lycos evaluation dataset. Model 1 achieves a five percent lower inverse harmonic mean rank than Model 0. However, the median rank of the correct document was substantially higher with Model 1.

For the sake of efficiency, the ranking algorithm used an inverted index, as described in the next section, to efficiently rank just those documents exhibiting some lexical overlap with the query (in the case of Model 0) or those documents containing a word w which is a high-probability replacement for a query word (in the case of Model 1). We will call the set of ranked documents the *qualifying set* for a query.

It may occasionally happen that the known relevant document for a query does not appear in qualifying set for that query. This could happen for a number of reasons, including

- The document (a web page, that is) may have changed between the time it was recorded in the Lycos transaction log and the time it was downloaded in preparation for training WEAVER. While the original document may have been relevant to the query, the updated document was not.
- The words shared by query and document were excised from the document during WEAVER's preprocessing, appearing, for instance, within an html `<meta>` element.
- The algorithm failed to recognize the relevancy relationship between the query and document.

In this case, the model assigns a score of zero for the correct document. We mark these queries as "defaulted" and exclude them from the cumulative results. Not surprisingly, the number of defaulted queries in Model 1 was significantly lower than that of Model 0. This discrepancy probably represents an unfair advantage for Model 0, which faced fewer difficult queries than Model 1.

queries	5321	
queries processed	4363	
documents ranked	624,491	
	Model 0	**Model 1**
model weights (α, β, γ)	$0, 0.99, 0.01$	$0.4, 0.594, 0.006$
defaulted queries	802	549
inv. harmonic rank	31.47	29.85
median rank	2422	3562

Table 3.3: Results of Lycos document-ranking experiments. The experiments involved ordering the 4363 test documents by relevance to each of the 4363 queries. The only known relevance judgments is a single query-document pairing according to the clickthrough data. The harmonic rank and median rank measure, in different ways, how highly the automatic ranking algorithm listed the matching document for each query.

3.5.3 Email data

In this section we explore the use of statistical document ranking techniques for the purpose of organizing email messages. More specifically, the goal was to explore the potential for learning a correlation between the subject line of an email (acting as a query) and its body (acting as a document). A system which could correlate message bodies with subjects accurately could conceivably be applied to the automatic categorization of emails, a task of great import not only to individuals with an unwieldy amount of email, but also to corporate call centers which could exploit such a system to assign incoming requests for help to different topics, priority levels, or experts.

The corpus contained 5731 documents: each document consisted of a subject line and email body; these emails were randomly sampled from a collection of personal correspondences accumulated by a single person over the span of three years.

A collection of email correspondences has very different characteristics than one consisting of newswire articles. Some IR research has investigated the task of online classification of email by content [51], but there has been scant work on searching and ranking emails within a document retrieval setting. For sure, the dearth of research arises from the difficulty inherent in procuring a distributable collection of email correspondences.

Specifically, the evaluation task was as follows.

- Provide a retrieval system with a randomly selected 95% portion of the (subject/body) pairs. The system can, in the case of *tfidf*, use this data to estimate term frequencies, or, in the case of the Model 1 system, construct a statistical word-relation model.
- Use the remaining five percent of the subject/body pairs to evaluate the system by ranking each body by relevance to each subject and calculating the average rank assigned to the correct body for each subject.

Table 3.4 summarizes the results of five comparisons between *tfidf* and Model 1, where each trial consisted of an independent, randomly-selected 95 : 5 partition of the email collection (in other words, each email record in the collection was assigned to the "training" category with probability 0.95 in each trial).

To illustrate the type of information contained in the distillation model, Table 3.5 shows four entries from the model.

Trial	# queries	arithmetic mean	inv. harmonic mean	# correct
1	305	73.7/37.3	1.92/1.86	138/141
2	294	73.1/35.7	1.82/1.70	140/154
3	264	73.5/32.9	1.93/1.84	114/120
4	294	77.5/38.0	1.94/1.78	129/145
5	279	73.7/36.4	1.91/1.76	123/139
Average:		74.3/36.0	1.90/1.78	128.8/139.8

Table 3.4: Comparing *tfidf* with Model 1 for retrieving emails by subject line. The values should be read as *tfidf*/Model 1. The last three columns are three different ways to gauge the quality of the ranking algorithm.

`copy`:	`copy` 0.985	`carbon` 0.003	`blind` 0.002
`ascii`:	`ascii` 0.438	`charset` 0.131	`text/plain` 0.126
`flight`:	`flight` 0.980	`airport` 0.007	`visit` 0.001
`at&t`:	`at&t` 0.952	`labs` 0.012	`research` 0.006

Table 3.5: The (tail-truncated) distributions for a select group of words. The distributions were learned from a collection of personal emails.

3.5.4 Comparison to standard vector-space techniques

The Model-0 (LM-based) and *tfidf*-based document ranking methods share the same weakness: an inability to account for word relatedness effects intrinsically. "Model 1"-style document ranking accounts for this shortcoming in LM-based retrieval, and query expansion addresses the same problem in *tfidf*-based retrieval.

Query expansion techniques such as Rocchio [75] use the original query to rank documents tentatively, and then expand the query with those words appearing most often in the highest ranked documents. Document ranking thus becomes a two-pass process. In contrast, Model 1 builds (offline) a statistical model of word-relatedness from the document corpus and uses that model in gauging relevance.

So Model 1 and query expansion are similar, in that they both rely on lexical co-occurrence statistics to handle word-relatedness. But these techniques differ in what data they mine for co-occurrence statistics and how they use that data: Model 1 examines the corpus as a whole, whereas query expansion examines documents related (via a high *tfidf* score) to the query.

This section reports on a brief empirical study of the relative behavior of these four

method	capsule summary
tfidf	cosine-based similiarity metric between document and query, where words are weighted according to their "information content."
tfidf + query expansion	words appearing relatively more frequently in the highest-ranked documents in a *tfidf* ranking are accorded a higher weight in a new, synthetic query, which is used in a second ranking.
Model 0	Uses a stochastic model constructed from the document (interpolated with a model constructed from the entire corpus) to "predict" the query. Documents whose models predict the query with high likelihood are accorded higher relevance.
Model 1	words related to those appearing in the query participate through a statistical model of word-relatedness. The model is calculated offline, independently of any particular query.

Figure 3.11: Capsule summary of four ranking techniques

algorithms—Model 0, Model 1, *tfidf* and *tfidf* with Rocchio-based query expansion—on a single task: TREC-style document ranking using newswire documents. For reference, Figure 3.11 contains a capsule summary of the four techniques.

We used the TREC AP88 corpus: 79,919 Associated Press newsfeed documents from 1988, with TREC topics (queries) 251-300 and TREC-supplied relevance judgments. The experiments reported here used only the title field from the topics. This dataset is somewhat atypical of traditional IR datasets in that relevant documents are rather sparse. In fact, among the 50 topics, two topics contained only one document judged relevant by the human assessors, and none of the four algorithms below placed that document among the 1000 documents deemed most relevant to the topic. Removing these two topics, as the four experiments reported below all did, reduces the number of topics to 48. For illustration, Figure 3.12 depicts one of these topics.

We begin with a brief description of the experimental procedure followed in each case, followed by comparative results and a discussion of those results. In general, the idea was to give each method the fullest opportunity to excel—by varying the appropriate parameters from each method and selecting the configuration which performed best on the evaluation data.

1. ***tfidf*-based ranking**: This experiment used the same *tfidf* ranking formula as else-

Topic: `Cigarette Consumption`

Description: `What data is available on cigarette consumption by country?`

Narrative: `If cigarette smoking is a causative factor in lung cancer, then countries with higher cigarette consumption per capita might experience a higher incidence of lung cancer. This topic would provide basic data for such a comparison.`

Normalized: `cigarett consumpt data cigarett consumpt countri cigarett smoke caus factor lung cancer countri higher cigarett consumpt capita experi higher incid lung cancer topic provid basic data comparison`

Figure 3.12: For reference, a representative topic from those used in the experiments of Section 3.5.4. The experiments reported in this section used the entire topic when adjudicating relevance. The "Normalized" entry refers to the view of this topic after converting the words to uppercase, stemming, and removing stopwords.

where in this section to rank the documents in the AP88 corpus by relevance to the provided topics [67].

2. *tfidf* **with Rocchio-based query expansion**: To implement query expansion, we employed a popular IR technique known as the Rocchio method. For a given query, the Rocchio-based ranking procedure works as follows:

   1) Rank documents using *tfidf*, as above.
   2) Take the top n_1 most relevant documents $\{r_1, r_2, \ldots r_{n_1}\}$ according to this ranking, and expand the query as follows:

$$\mathbf{q} \leftarrow \mathbf{q} + \beta \sum_{i=1}^{n_1} \frac{r_i}{n_1} \tag{3.18}$$

   3) Rerank documents with respect to this updated (expanded) topic.

   In general, the Rocchio method involves an additive *and* subtractive term:

$$\mathbf{q} \leftarrow \mathbf{q} + \beta \sum_{i=1}^{n} \frac{r_i}{n_1} - \gamma \sum_{i=1}^{n_2} \frac{r_i}{n_2} \tag{3.19}$$

cigarett (7.9375) cancer (3.95) smoke (3.8875) consumpt (3.3375) lung (3.0875) tobacco (2.5875) higher (2.375) data (2.15) number (2.0625) countri (2.0375) topic (2) year (1.725) death (1.425) smoker (1.275) percent (1.275) caus (1.15) basic (1.1125) provid (1.075) capita (1.075) incid (1.0375) factor (1.0375) experi (1.0375) comparison (1.0375) _ (0.9375) cipollon (0.8625) compani (0.7875) report (0.75) case (0.7125) american (0.7125) product (0.675) rate (0.6) health (0.6) billion (0.6) societi (0.5625) research (0.5625) monei (0.5625) state (0.525) feder (0.525) cost (0.525) mr (0.4875) leukemia (0.4875) danger (0.4875) claim (0.4875) women (0.45) warn (0.45) 1987 (0.45) lawyer (0.4125) evid (0.4125) 10 (0.4125) studi (0.375) morri (0.375) maker (0.375) link (0.375) increas (0.375) group (0.375) gener (0.375) drop (0.375) diseas (0.375) dai (0.375) attornei (0.375) price (0.3375) market (0.3375) liabil (0.3375) gunsalu (0.3375) fda (0.3)

Figure 3.13: An expanded version of the topic from Figure 3.12, using Rocchio to estimate the weights.

In these experiments, we took $\beta = 0.75$ (a somewhat standard value in the industry), and $\gamma = 0$, effectively deactivating the subtractive term. We found that taking $n_1 = 10$ performed best, though in fact this value is rather lower than the "industry standard" range of between 30 and 50.

As the graph in Figure 3.14 shows (and as is expected), query expansion provides an improvement over "vanilla" *tfidf* except at the very highest precision range.

As an illustration, Figure 3.13 displays the top few terms, along with their weights, for the query-expanded version of the topic in Figure 3.12.

3. **Model-0 results**: In these experiments, we found that setting $\alpha = 0.80$ was optimal. The language modelling approach performs surprisingly well relative to traditional vector-space techniques, though it cannot match the performance of *tfidf* with query expansion at the lower-precision regions.

4. **Model-1 results**

   As we have described earlier, Model 1 includes a word-to-word statistical relation model, which effectively spreads the probability of a word over a number of related concepts. Two examples of individual word transition vectors are:

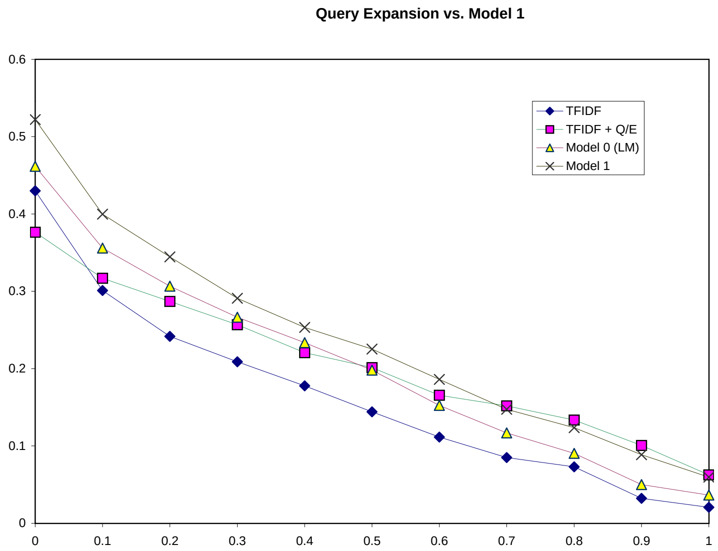


Figure 3.14: Precision-recall curves for four techniques under discussion in this section: *tfidf*, *tfidf* with Rocchio-based query expansion, Model 0 and Model 1.

`cigarett`: `cigarett` 0.268101 `smoke` 0.0510487 `tobacco` 0.0369069 `smoker` 0.0273317 `pack` 0.0181277 `brand` 0.00906122 `rj` 0.00877782 `lung` 0.00708167 `carton` 0.00667786 `product` 0.00616576

`cancer`: `cancer` 0.242286 `breast` 0.0113614 `diseas` 0.0101922 `studi` 0.00693828 `treatment` 0.00580977 `lung` 0.00568525 `evid` 0.00508431 `tumor` 0.00505677 `surgeri` 0.00501926 `smoke` 0.0043776

Using these models and (3.17), a Model 1-based ranking algorithm exhibited a performance superior to *tfidf* with Rocchio-based query expansion, as seen in Figure 3.14—except at the lowest (and usually least interesting) part of the curve.

3.6 Practical considerations

Conventional high-performance retrieval systems typically decompose the task of ranking documents by relevance to a query $\mathbf{q} = \{q_1, q_2, \ldots q_n\}$ into a retrieval and a query expansion stage. For instance, in the automatic relevance feedback approach, the system first ranks just those documents whose content overlaps with $\mathbf{q}$, and assumes the other members of $\mathcal{D}$ are not relevant to the query. In part because there may exist relevant documents which have no words in common with the query, the system then expands $\mathbf{q}$ to include a set of words which appeared frequently among the top-ranked documents in the first step, and ranks documents whose content overlaps with this expanded query.

The crucial aspect of this two-step process is that in each step, the ranking algorithm can disregard documents containing none of the words in the query. This approximation—ignoring the potentially large subset of the document collection with no words in common with the query—makes the difference between a usable and impractically slow ranking algorithm.

A retrieval system that only considers documents containing words in the query can organize the collection efficiently into an inverted index, which lists for each word the documents containing that word. Processing a query with an inverted index is then a simple matter of visiting just those lists in the inverted index corresponding to words in the query. Inverted indices are a nearly ubiquitous component of large-scale document retrieval systems.

At first glance, it would seem that models which capture the semantic proximity between words are incompatible with the use of an inverted index. After all, when using Model 1 or Model 1', all documents are "in play" for a given query: a document not containing a query word might, after all, still generate that word with high probability. But forfeiting the use of an inverted index entirely and explicitly computing a relevance score for every document, as NaiveRank does, is too inefficient. Calculating the relevance of a document to a query using Model 1 (equation (3.13)) requires time proportional to $|\mathbf{q}| \times |\mathbf{d}|$: the product of the size of the query and the size of the document. In practice, it appears that NaiveRank can require *an hour or more per query* for a TREC-sized document collection of a few hundred thousand documents, on a modern-day workstation.

The remainder of this section presents a set of heuristics to allow efficient (albeit approximate) ranking of documents by their probability $p_\alpha(\mathbf{q} \mid \mathbf{d})$ of generating the query in a distillation process. This section also shows how, by using a data structure similar to an inverted index, one can achieve near real-time retrieval performance.

The key observation is that the ranking algorithm offers a time-space tradeoff. Rather than calculating the sum in (3.13) during ranking, one can precompute $p(q \mid \mathbf{d})$ for every known word q and each document $\mathbf{d} \in \mathcal{D}$, and store the results in a matrix, illustrated in Figure 3.6. Denote this "inverted matrix"—similar to an inverted index, but containing an entry for *every* (d, q) pair—by the symbol $\mathcal{I}$. (As a reminder: $p(q \mid \mathbf{d})$ is just one component

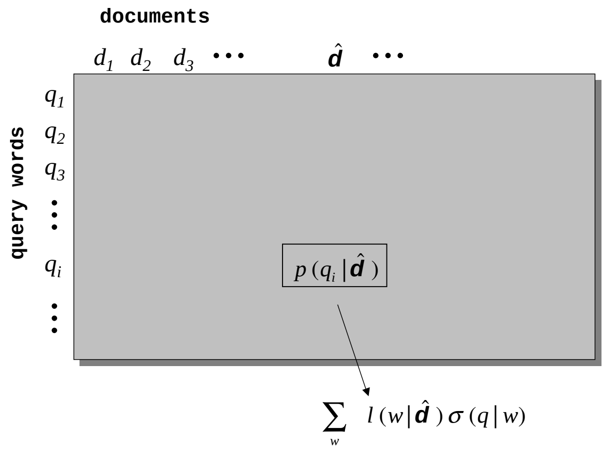


Figure 3.15: NAIVERANK computes $p_\alpha(q_i \mid \mathbf{d})$ according to (3.16) for each word q_i in the query $\mathbf{q} = \{q_1, q_2 \ldots q_m\}$. Avoiding this costly process is not difficult: just precompute, once and for all, $p(q \mid \mathbf{d})$ for all words q and documents d. Calculating $p_\alpha(\mathbf{q} \mid \mathbf{d})$ is then a matter of multiplying the precomputed $p(\mathbf{q}_i \mid \mathbf{d})$ together, factoring in the smoothing terms $p(q \mid \mathcal{D})$ along the way. This figure depicts a data structure $\mathcal{I}$ which stores these precomputed values.

of the smoothed probability $p_\alpha(q \mid \mathbf{d})$ of q given $\mathbf{d}$. By inspection of (3.16), one can see also a contribution from the document-wide language model $p(q \mid \mathcal{D})$.)

Precomputing the cells of $\mathcal{I}$ and then using these values in NAIVERANK reduces the cost of ranking from $\mid \mathcal{D} \mid \times \mid \mathbf{q} \mid \times \mid \mathbf{d} \mid$ to $\mid \mathcal{D} \mid \times \mid \mathbf{q} \mid$ operations.

Unfortunately, the matrix $\mathcal{I}$, with as many columns as documents in the collection and as many rows as there are distinct words recognized by the system, can be prohibitively expensive to compute and store. A $100,000$ document collection and $100,000$ word vocabulary would require a matrix $400GB$ in size. One can therefore make the following approximation to (3.16):

$$p_\alpha(q \mid \mathbf{d}) \approx \alpha\, p(q \mid \mathcal{D}) + (1-\alpha) \sum_{w \in \mathcal{T}^n(q)} l(w \mid \mathbf{d})\, \sigma(q \mid w) \qquad (3.20)$$

$$\text{where} \quad \mathcal{T}^n(q) \stackrel{\text{def}}{=} \{w : \sigma(q \mid w) \ \text{ is among the } n \text{ largest } \sigma\text{-values for any } w\}$$

Roughly speaking, $T^n(q)$ is the set of n words most likely to map to q. In other words, (3.20) assumes that each document covers at most n concepts. In the performed experiments, n was set to 25. Making this approximation results in most values $p(q \mid \mathbf{d})$ dropping to zero, yielding a sparse $\mathcal{I}$ matrix—easy to store and precompute using conventional sparse-matrix techniques.

Of course, any approximation runs the risk of gaining speed at the cost of accuracy. To address this, the new ranking algorithm therefore rescores (and reranks) the top-scoring documents according to (3.16).

Algorithm 6: *"FastRank": efficient document ranking*

Input: `Query` $\mathbf{q} = \{q_1, q_2, \dots q_m\}$;
`Collection of documents` $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \dots \mathbf{d}_N\}$;
`Word-relation probability` $\sigma(q \mid w)$ `for all word pairs` q, w
`Inverted mapping` ψ `from words to documents`

Output: `Relevance score` $\rho_{\mathbf{q}}(\mathbf{d})$ `for each document` $\mathbf{d}$

1. `Do for each document` $\mathbf{d} \in \mathcal{D}$ `in the collection`
2. `Set` $\rho_{\mathbf{q}}(\mathbf{d}) \leftarrow 1$
3. `Do for each query word` $q \in \mathbf{q}$
4. `Do for each document` $\mathbf{d} \in \mathcal{D}$
5. `Set` $p_\alpha(q \mid \mathbf{d}) \leftarrow \alpha p(q \mid \mathcal{D})$ `(precomputed)`
6. `If` $\mathbf{d} \in \mathcal{I}(\mathbf{q})$ `then` $p_\alpha(q \mid \mathbf{d}) \leftarrow p_\alpha(q \mid \mathbf{d}) + (1-\alpha) p(q \mid \mathbf{d})$ `(precomputed)`
7. `Rescore the top-ranking documents according to (3.16).`

Figure 3.16 shows that, on the AP subset of the TREC dataset, the precision/recall performance of the fast, approximate algorithm is essentially indistinguishable from the naive, exact algorithm. But the former algorithm is considerably faster: on a 266Mhz workstation with 1.5GB of physical memory, NaiveRank required over an hour per query while FastRank required an average of only 12 seconds per query[2].

3.7 Application: Multilingual retrieval

In many real-world settings (such as the Internet), several different languages may appear within a collection. Ideally, a document retrieval system should be capable of retrieving a document relevant to the user's query no matter what the language of the document.

[2]Had the time difference between FastRank and NaiveRank been less marked, one might reasonably insist on a more rigorous evaluation framework: running the system in single-user mode, clearing the I/O caches, running multiple trials, and so forth.

	NaiveRank	FastRank
Relevant:	5845	5845
Rel.ret.:	4513	4386
Precision:		
at 0.00	0.7411	0.7411
at 0.10	0.5993	0.5994
at 0.20	0.5291	0.5295
at 0.30	0.4487	0.4501
at 0.40	0.4079	0.4085
at 0.50	0.3646	0.3599
at 0.60	0.3125	0.3161
at 0.70	0.2721	0.2698
at 0.80	0.2136	0.2043
at 0.90	0.1366	0.1433
at 1.00	0.0353	0.0339
Avg.:	0.3531	0.3523
Precision at:		
5 docs:	0.5489	0.5489
10 docs:	0.5255	0.5255
15 docs:	0.5149	0.5163
20 docs:	0.4883	0.4883
30 docs:	0.4553	0.4567
100 docs:	0.3351	0.3357
200 docs:	0.2596	0.2601
500 docs:	0.1582	0.1594
1000 docs:	0.0960	0.0933
R-Precision:	0.3707	0.3718

Figure 3.16: The NaiveRank and FastRank algorithms yield almost indistinguishable results when applied to the AP portion of the TREC data using the narrative fields of queries 51-100.

To simplify the exposition, focus on a two-language scenario: a user issues a query $\mathbf{q}$ in a source language $\mathcal{S}$, and the system ranks documents in a target language $\mathcal{T}$ by their relevance to $\mathbf{q}$.

Some popular strategies for this problem are:

1. Cast the problem as a monolingual one by translating the documents into language $\mathcal{S}$. The cost of translating the entire collection $\mathcal{D}$ of documents may be expensive, but the computation can be performed offline, prior to processing a user's query.

2. Cast the problem as a monolingual one by translating the query to the language $\mathcal{T}$ [64]. This obviates the need to translate the entire collection, which can be an expensive proposition. However, since queries are typically quite short, accurately translating the query may be impossible; the context of a word (its neighbors) usually plays an important role in disambiguating the word's meaning during translation. For instance, should a multilingual retrieval system render the English query `suit` into the French query `costume` (piece of clothing) or into `procès` (legal action)? In principle, translating words within a document can be easier because surrounding words often serve to disambiguate the meaning.

3. Franz *et al.* have also suggested a hybrid approach which ranks documents in two different ways: first, by translating the query into $\mathcal{T}$, and second, by translating the documents into $\mathcal{S}$. They have demonstrated that the performance of a system which employs *both* techniques can exceed that resulting from the application of either strategy alone [31].

Another approach—the one pursued here—is to avoid explicit translation altogether, by incorporating translation into the retrieval framework. In other words, perform translation and retrieval simultaneously. This idea has precedent in the IR literature: Dumais *et al.* have, for instance, proposed the use of latent semantic indexing to perform retrieval across multiple languages [25].

Performing retrieval across languages within the framework described in Section 3.3 is a straightforward matter. One can use model (3.12) as before, but now interpret $\sigma(w \mid q)$ as a measure of the likelihood that a word q in the language $\mathcal{S}$ is a translation of a word w in the language $\mathcal{T}$. In other words, σ is now a model of *translation* rather than semantic proximity.

Presented with a bilingual collection of $(\mathbf{q}, \mathbf{d})$ pairs, where each $\mathbf{q}$ is in the language $\mathcal{S}$ and each $\mathbf{d}$ is in $\mathcal{T}$, applying the EM-based strategy of Section 3.4 would work without modification. Nothing in the models described in Section 3.3 assumes the queries and documents are in the same language.

3.8 Application: Answer-finding

Searching the web or skimming a lengthy manual to find the answer to a specific question can be a tedious exercise. Moreover, for a large retail company, employing a battalion of customer-support personnel to perform this same task on behalf of telephone customers can be an expensive proposition. A recent study has concluded that providing help to a single customer via a live telephone operator can cost a company $20 to $25 per call [32]. This section investigates how the statistical techniques of this chapter can help in automating the process of answer-finding. The ultimate goal is a system which, equipped with a large collection of prepackaged answers, can automatically identify the best response to a user's query.

Starting from a large collection of answered questions, the algorithms described here learn lexical correlations between questions and answers. Two examples of such correlations are

- Questions containing the word `why` are more likely, in general, to be paired with an answer beginning with the word `because`.
- A question containing the word `vacation` is likely to be paired with an answer containing one of the words {`flight, trip, cruise`}.

To serve as a collection of answered questions, this section relies on two types of datasets:

Usenet FAQs: A collection of Usenet frequently-asked question (FAQ) documents. This dataset, a dynamic and publically available entity[3], presently contains several thousand individual FAQ documents, totalling hundreds of megabytes. The topics of these documents range from libertarianism to livestock predators to Fortran programming. This section uses, for experimental purposes, a set of 200 documents from the `comp.*` Usenet hierarchy containing 1800 questions.

Call-center dialogues: A collection of questions submitted by customers to Ben & Jerrys, along with the answer supplied by a company representative. This dataset contained 5145 question/answer pairs.

One can cast answer-finding as a traditional document retrieval problem by considering each answer as an isolated document and viewing the query as just another (albeit smaller)

[3] The Usenet FAQ collection is available at `ftp://rtfm.mit.edu` and `http://www.faqs.org`. The Ben & Jerrys dataset is proprietary.

What is ViewKit? Is there a free version?

ViewKit is an enhanced version of the C++/Motif framework that ...

What's the difference between strong AI and weak AI?

Strong AI makes the bold claim that computer can be made to think on a level (at least) equal to humans …

How can I get my cisco to talk to a third-party router over a serial link?

You need to tell your cisco to use the same link-level protocol as the other router; by default, ciscos ...

Do you use only free range dairy?

Hello. Well, other than putting up fences so they won't wander to East Pakistan, ...

Do you still sell Chubby Hubby T Shirts? If so, where can I find one? Thanks.

Hello. We no longer sell Chubby Hubby T Shirts. Sorry. To see what we do sell, ...

I have been reading about your new smoothie product for six months or so. Is it being distributed anywhere in Lincoln NE?

We are sorry to tell you that our frozen smothies have no distribution in Nebraska...

Figure 3.17: Excerpts from two of the question/answer corpora used here. *Left*: Q/A pairs from the Usenet `comp.*` newsgroups. *Right*: Q/A pairs from Ben & Jerry's customer support.

document. Traditional *tfidf*-based ranking of answers will reward candidate answers with many words in common with the query.

Employing traditional *tfidf*-based vector-space retrieval to find answers seems attractive, since *tfidf* is a standard, time-tested algorithm in the toolbox of any IR professional. However, the experiments reported below demonstrate that standard *tfidf* retrieval performs poorly compared with techniques that "learn" to locate answers by inspection of a collection of answered questions.

The lexical chasm

In ranking documents by relevance to a query, traditional information retrieval systems place a large emphasis on lexical similarity between document and query: the closer the distribution of words in a candidate document is to the query, the more relevant is the question. Many users of document retrieval systems have this model (or some vague notion of it) in mind, and in formulating their query they usually employ terms that they expect would appear in a relevant document. But users who submit questions to an answer-finding system can't be expected to anticipate the lexical content of an optimal response: there is often very little overlap between the terms in a question and the terms appearing in its answer. For example, the best response to the question `Where's a good place to get dinner?` might be `Zaphod's Bar and Grill has great fajitas`, which have *no* tokens in common.

More generally, questions often contain terms that differ from, but are related to, the terms in the matching answer. The group of terms {`what`, `when`, `where`, `why`, `how`} will typically appear more frequently in questions than answers, for example. The legal vocabularies for questions and answers are the same, but the probability distributions over those vocabularies are different for questions and their answers.

Furthermore, the probability distribution for terms in the answer is linked to the probability distribution of the terms in the question. Thus there is both a mismatch between the terms in queries and the terms in responses matching those queries, as well as a correspondence between the mismatched terms in the query and response. For example, in a `where` question, the response frequently contains the words {`near, adjacent, street, on`} and so forth.

This combination of a vocabulary mismatch and linkage between query and response vocabularies is in some sense a *lexical chasm*. The query is on one side of the chasm and the response on the other side. The vocabularies on the two sides of the chasm are the same, but the distributions differ on each side of the chasm. The distributions on the two sides of the chasm are linked at the semantic and discourse levels.

This chasm suggests that traditional bag-of-words retrieval might be less effective at matching questions to responses than matching keywords to documents. To bridge the lexical chasm, an IR system must adopt a strategy that rises from the lexical level towards the semantic level.

Traditional IR systems based on the *tfidf* ranking criterion [76] suffer from a particular form of the lexical gap problem, namely the problem of synonymy. A query containing the term `Constantinople` ought to fetch documents about Istanbul, but doing so requires a step beyond comparing the word frequency histograms in query and candidate documents. The techniques introduced in this chapter are designed to bridge the lexical gap between questions and answers by characterizing the co-occurrence between one word in a query and another word in an answer. Of course, traditional vector-space document ranking methods address the lexical mismatch problem as well, using query expansion.

When there's only one known relevant document for each query (as is the case here), What really counts is how close a correct answer is to the top of the returned list. Instead of precision-recall measures, therefore, this section uses the rank of the correct answer in the returned list as a metric of performance. More specifically, it relies in inverse harmonic mean rank.

The advantage of the median is that it is less affected by non-representative tails of the distribution. The inverse harmonic mean rank is designed to give an intuitive feel for where the correct answer is likely to appear. It also penalizes rank changes near the top more than

Usenet `comp.*` FAQs

Method	Median	p	Inv. Harmonic Mean	p
tfidf	3.0 0	-	4.12	-
translation	1.60	0.008	1.72	<0.001

Ben & Jerry's Call Center FAQ

Method	Median	p	Inv. Harmonic Mean	p
tfidf	16.6	-	6.25	-
translation	25.2	-	3.41	<0.001

Table 3.6: Answer-finding experiments on Usenet, and a call-center dataset. The numbers here are averaged over five runs of randomly selected testing set of 10% of the document sets. The p values are unpaired t-statistics for the test that the model outperforms the baseline.

changes farther away; a drop in rank from two to three is more significant than a change from 99 to 100.

Experiments

The framework introduced in this chapter applies to question-answering as follows. One can equate the relevance of an answer $\mathbf{r}$ to a question $\mathbf{q}$ with the quantity $p(\mathbf{q} \mid \mathbf{r})$. The entries of the stochastic "word-relatedness" matrix in this case have the interpretation that the i, jth cell reflects the likelihood that an answer containing the word j corresponds to a question containing word i.

There are reasons to think an approach inspired by language translation might work well for question-answering: trained on a sufficient amount of question/answer pairs, the translation model should learn how answer-words "translate to" question-words, bridging the lexical chasm. For instance, words like `at, location, place, street, directions` will all translate with reasonably high probability to the question-word `where`.

Using an alignment a between question and answer words, $p(\mathbf{q} \mid \mathbf{r})$ decomposes as

$$p(\mathbf{q} \mid \mathbf{r}) = \sum_a p(\mathbf{q}, a \mid \mathbf{r}) = \sum_a p(\mathbf{q} \mid a, \mathbf{r}) p(a \mid \mathbf{r}) \tag{3.21}$$

From here one can follow the derivation following (3.7), with $\mathbf{d}$ now replaced by $\mathbf{r}$.

Having learned the word-to-word synonymy parameters from the training data, the system is then ready to perform answer-finding as follows. Starting from an input question $\mathbf{q}$, rank each answer according to $p(\mathbf{q} \mid \mathbf{r})$ via (3.9). The sum in (3.9) is over an exponential number of alignments, but one can calculate this value efficiently by rearranging the sum and product—by now a familiar routine.

As Table 3.6 indicates, using WEAVER for answer-finding has promise. The only exception is the median rank on the Ben & Jerry's problem. Interestingly, while the median rank falls, the harmonic mean rises considerably. The inverse harmonic mean is more sensitive to smaller numbers (documents with higher rank). This suggests a higher fraction of correct documents ranked close to the top than with *tfidf*—the behavior one would expect from an answer-finding system.

Extensions

Exploiting document structure

The experiments reported in this section treat the question/answer pairs as isolated objects. In reality, they often occur as part of a larger document structure. There are several strategies for exploiting this structure to improve the accuracy of answer retrieval.

One idea is to try to find not the individual answer best matching the input question, but instead the best *region*—collection of answers, say—for the question. Giving a user a larger body of text which probably contains the correct answer is of course inferior to providing just the answer, but better than providing the wrong answer, or no answer at all. The IR community has explicitly acknowledged this multi-level definition of correctness; in the TREC question-answering track, systems may participate in the 55-byte or 255-byte subtracks. In the former, participating systems must identify a window of at most 55 words containing the answer; in the latter, systems are permitted up to 255-word windows [84].

Another approach is to introduce a function of the position of an answer in an FAQ as a prior probability that the answer is appropriate. It may be, for example, that simpler, more general questions usually occur early in a user's manual, and people generally ask more general questions first; an obvious strategy in this case would be to bias towards the first several answers early in a dialogue with a user.

Exploiting question structure

Questions come in different flavors: `who`-type questions are characterized by a somewhat different syntax and lexicon than `where`-type questions. Answers to these questions are different as well. For instance, words and phrases like `behind`, `next to`, and `near` may appear with a higher frequency in answers to `where`-questions than answers to `who`-questions.

We have already discussed how Model 1 inherently accounts for the "lexical mismatch" between questions and answers. Going further, however, one could try to exploit the difference between question types; accounting, for instance, for the difference between answers to `where` questions and answers to `who` questions. The central idea is to automatically identify the type (who, what, why, where, how) of an input question, and use that information to help assess candidate answers. Ideally, the resulting algorithm would bias the candidate answers in favor of `where` answers when processing an `where` question.

One way to incorporate such a change into a probabilistic framework is as follows. Introduce a class-based model $p(t_{\mathbf{q}} \mid t_{\mathbf{r}})$, which assigns a probability to the event that an answer of type $t_{\mathbf{r}}$ is the correct response to a question of type $t_{\mathbf{q}}$. Also introduce a model $p(\mathbf{q} \mid t_{\mathbf{q}})$ for generating a query $\mathbf{q}$ from a query class $t_{\mathbf{q}}$. Interpolating this model with (3.7) gives

$$p(\mathbf{q} \mid \mathbf{r}) = \alpha p(t_{\mathbf{q}} \mid t_{\mathbf{r}}) p(\mathbf{q} \mid t_{\mathbf{q}}) + (1-\alpha) \sum_a p(\mathbf{q}, a \mid \mathbf{r}) \tag{3.22}$$

Unfortunately, the problem of identifying question type has long been recognized as difficult. [50]. For instance, the question "How do I get to the World Trade Center" appears to be a `how`-question, but is implicitly more of a `where`-question.

* * *

This section focuses on the task of locating an answer within a large collection of candidate answers. This is to be contrasted with the problem of *question-answering*, a considerably more ambitious endeavor, requiring the construction of an answer by searching a large collection of text. Question answering systems are usually domain-specific and highly knowledge-intensive, applying sophisticated linguistic analysis to both the question and the text to be searched for an answer.

Somewhat more closely related to the present work is the FAQ-FINDER system under development at U.C. Irvine [16]. The system attempts to locate, within a collection of Usenet FAQ documents, the most appropriate answer to an input question. The FAQ-FINDER system is similar to the work described in this paper in starting with a *tfidf*-based answer-scoring approach. In trying to bridge the lexical chasm, however, the paths diverge: FAQ-FINDER relies on a semantic network to establish correlations between related terms such as `husband` and `spouse`. In contrast, the WEAVER approach depend only on the availability of a suitable training set. By not relying on any external source of data, Model 1 appears to be better suited to the production of "vertical" document ranking applications.

That is, given a collection of medical documents, say, or legal or financial documents, the techniques described in this chapter describe how to construct from these documents, with no additional data gathering or human annotating, a domain-specific text ranking system with an intrinsic notion of query expansion.

3.9 Chapter summary

Taking as a starting point the idea of using statistical language models for document retrieval, this chapter has demonstrated how machine learning techniques can give rise to more sophisticated and powerful models not only for document retrieval, but also for a wide range of problems in information processing.

After outlining the approach, this chapter presented two closely related models of the document-query "translation" process. With the EM algorithm, the parameters of these models can be learned automatically from a collection of documents. Experiments on TREC data, user transactions from a large web search engine, and a collection of emails demonstrate that even these simple methods are competitive with standard baseline vector space methods. In some sense, the statistical word-relatedness models introduced here are theoretically principled alternatives to query expansion.

Of course, the actual models proposed here only begin to tap the potential of this approach. More powerful models of the query generation process should offer performance gains, including:

> *Explicit fertility models*: One of the fundamental notions of statistical translation is the idea of *fertility*, where a source word can generate zero or more words in the target sentence. While there appears to be no good reason why a word selected from the document should generate more than a single query term, one might benefit from the added sophistication of a model which recognizes *infertility* probabilities: some words or phrases are more likely than others to generate no terms at all in the query. For instance, the phrases `This document is about` or `In conclusion` carry negligeable information content. The use of stop word lists mitigates but does not eliminate the need for this feature.
>
> *Discarding the independence assumption*: WEAVER makes the usual "bag of words" assumption about documents, ignoring word order in the source document for the sake of simplicity and computational ease. But the relative ordering of words is informative in almost all applications, and crucial in some. The sense of a word is often revealed by nearby words, and so by heeding contextual clues, one might hope to obtain a more accurate mapping from document words to query words.

Recognizing word position within a document: In most cases, the beginning of a document is more important, for the purposes of distillation, than the end of that document. Someone looking for information on Caribbean vacations would typically prefer a document which covers this topic in the very beginning over a document which does not mention it in the first hundred page. The proposed models do not recognize this distinction, but one could imagine biasing the $l(\cdot \mid \mathbf{d})$ distribution to accord lower weight to words appearing near the end of a document. Of course, this feature is a special case of the previous one.

Chapter 4

Document gisting

This chapter introduces OCELOT*, a prototype system for automatically generating the "gist" of a web page by summarizing it. Although most text summarization research to date has focused on the task of news articles, web pages are quite different in both structure and content. Instead of coherent text with a well-defined discourse structure, they are more often likely to be a chaotic jumble of phrases, links, graphics and formatting commands. Such text provides little foothold for extractive summarization techniques, which attempt to generate a summary of a document by excerpting a contiguous, coherent span of text from it. This chapter builds upon recent work in* non-extractive *summarization, producing the gist of a web page by "translating" it into a more concise representation rather than attempting to extract a representative text span verbatim.* OCELOT *uses probabilistic models to guide it in selecting and ordering words into a gist. This chapter describes a technique for learning these models automatically from a collection of human-summarized web pages.*

4.1 Introduction

The problem of *automatic text summarization* is to design an algorithm to produce useful and readable summaries of documents without human intervention. Even if this problem were well-defined (which it is not), it appears to be profoundly difficult. After all, humans engaged in the summarization task leverage a deep semantic understanding of the document to be condensed, a level of analysis well beyond the reach of automation using current technology.

An important distinction in summarization is between generic summaries, which capture the central ideas of the document in much the same way that the abstract of this chapter was designed to distill its salient points, and query-relevant summaries, which reflect the relevance of a document to a user-specified query. This chapter focuses on the generic summarization problem, while the following chapter looks at query-relevant summarization.

Since condensing a document into a useful and meaningful summary appears to require a level of intelligence not currently available in synthetic form, most previous work on summarization has focused on the rather less ambitious goal of *extractive summarization*: selecting text spans—either complete sentences or paragraphs—from the original document, and arranging the segments in some order to produce a summary. Unfortunately, this technique seems to be a poor fit for web pages, which often contain only disjointed text.

The OCELOT approach to web page summarization is to *synthesize* a summary, rather than extract one. OCELOT relies on a set of statistical models to guide its choice of words and how to arrange these words in a summary. The models themselves are built using standard machine learning algorithms, the input to which is a large collection of human-summarized web pages. Specifically, this chapter uses data from the Open Directory Project [66], a large and ongoing volunteer effort to collect and describe the "best" web sites on the Internet. As of January 2000, the Open Directory Project contained 868, 227 web pages, each annotated with a short (roughly 13 word) human-authored summary.

Some important prior work in extractive summarization has explored issues such as cue phrases [52], positional indicators [27], lexical occurrence statistics [59], and the use of implicit discourse structure [56]. Most of this work relies fundamentally on a property of the source text which web pages often lack: a coherent stream of text with a logical discourse structure. Somewhat closer in spirit to OCELOT is work on combining an information extraction phase followed by generation; for instance, the FRUMP system [23] used templates for both information extraction and presentation—but once again on news stories, not web pages.

The very notion that a generic web page summarizer would be useful is predicated, in a sense, on the laziness of web page authors. After all, HTML offers multiple opportunities to web page authors (the title field, for instance, and the `meta description` field) to include a summary of the page's contents. But neither of these fields is required by HTML, and even when present, their content is often only marginally informative. Lastly, query-relevant summaries (which are not the focus of this chapter) will always need to be generated dynamically anyway, since the query isn't known at the time the page is written.

The OCELOT project bears a close relation to the work on automatic translation of natural language described earlier. To review, the central idea of statistical machine translation is that starting from a "bilingual" corpus of text, one can apply statistical machine learning algorithms to estimate maximum-likelihood parameter values for a model of translation between the two languages. For instance, the Candide system at IBM [6] used the proceedings of the Canadian parliament—maintained in both French and English—to learn an English-French translation model. In an entirely analogous way, one can use Open Directory's "bilingual corpus" of web pages and their summaries to learn a mapping from web pages to summaries. Probably the fundamental difference between OCELOT's task and natural language translation is a degree of difficulty: a satisfactory translation of a sentence must capture its entire meaning, while a satisfactory summary is actually *expected* to leave out most of the source document's content.

Besides its pedigree in statistical machine translation, this work is most similar to the non-extractive summarization system proposed by Witbrock and Mittal [87] in the context of generating headlines automatically from news stories. It also bears some resemblance, in its use of probabilistic models for word relatedness, to recent work in document retrieval [7, 8].

4.2 Statistical gisting

Conceptually, the task of building the OCELOT system decomposes as follows: (a) *content selection*: determining which words should comprise the summary, (b) *word ordering*: arranging these words into a readable summary, and (c) *search*: finding that sequence of words which is optimal in the dual senses of content and readability.

Content Selection

This chapter proposes two methods for word selection. The simpler of the strategies is to select words according to the frequency of their appearance in the document $\mathbf{d}$. That is, if word w appears with frequency $\lambda(w \mid \mathbf{d})$ in $\mathbf{d}$, then it should appear in a gist $\mathbf{g}$ of that document with the same frequency:

$$E[\lambda(w \mid \mathbf{g})] = E[\lambda(w \mid \mathbf{d})].$$

Here $E[\cdot]$ is the expectation operator. This technique is essentially identical to the "language modelling approach" to document retrieval proposed recently by Ponte and Croft [68].

A natural extension is to allow words which do not appear in the document to appear in the gist. To do so, this chapter recycles the technique introduced in Chapter 3 for automatically discovering words with similar or related meaning.

Surface Realization

In general, the probability of a word appearing at a specific position in a gist depends on the previous words. If the word `platypus` already appeared in a summary, for instance, it's not likely to appear again. And although `the` might appear multiple times in a summary, it is unlikely to appear in position k if it appeared in position $k-1$. The gisting model which OCELOT uses takes into account the ordering of words in a candidate gist by using an n-gram model of language.

Search

Though the tasks of content selection and surface realization have been introduced separately, in practice OCELOT selects and arranges words simultaneously when constructing a summary. That is, the system produces a gist of a document $\mathbf{d}$ by searching over all candidates $\mathbf{g}$ to find that gist which maximizes the product of a content selection term and a surface realization term. OCELOT applies generic Viterbi search techniques to efficiently find a near-optimal summary [29].

4.3 Three models of gisting

This section introduces three increasingly sophisticated statistical models to generate the gist of a given document. The next section will include a discussion of how to estimate the parameters of these models.

The idea of viewing document gisting as a problem in probabilistic inference is not prevalent. But intuitively, one can justify this perspective as follows. To begin, postulate a probabilistic model $p(\mathbf{g} \mid \mathbf{d})$ which assigns a value (a probability) to the event that the string of words $\mathbf{g} = \{g_1, g_2, \ldots g_n\}$ is the best gist of the document $\mathbf{d} = \{d_1, d_2 \ldots d_m\}$. One way to think about such a model is as the limiting value of a hypothetical process. Give the document $\mathbf{d}$ to a large number of people and ask each to produce a gist of the document. The value $p(\mathbf{g} \mid \mathbf{d})$ is the fraction of participants who produce $\mathbf{g}$ as the number of participants goes to infinity.

Given a document $\mathbf{d}$, the optimal gist for that document is, in a maximum likelihood sense,

$$\mathbf{g}^{\star} \quad = \quad \arg\max_{\mathbf{g}} p(\mathbf{g} \mid \mathbf{d}). \tag{4.1}$$

This section hypothesizes a few forms of the model and applies traditional statistical methods—maximum-likelihood estimation and in particular the expectation-maximization (EM) algorithm—to compute the parameters of the hypothesized models.

I. A "bag of words" approach

According to this model, a person gisting a document $\mathbf{d}$ begins by selecting a length n for the summary according to some probability distribution ϕ over possible lengths. Then, for each of the n assigned positions in the gist, he draws a word at random, from the document to be gisted, and fills in the current slot in the gist with that word. In combinatorial terminology, the values of the words in the gist are *i.i.d.* variables: the result of n independently and identically distributed random trials. In imagining a person composes a gist in such a way, this model makes a strong independence assumption among the words in the input document, viewing them as an unordered collection.

Algorithm 7: *Bag of words gisting*

Input: `Document` $\mathbf{d}$ `with word distribution` $\lambda(\cdot \mid \mathbf{d})$`;`

`Distribution` ϕ `over gist lengths;`

Output: Gist $\mathbf{g}$ of $\mathbf{d}$

1. `Select a length` n `for the gist:` $n \sim \phi$
2. `Do for` $i = 1$ `to` n
3. `Pick a word from the document:` $w \sim \lambda(\cdot \mid \mathbf{d})$
4. `Set` $g_i = w$

Once again denoting the frequency of word w in $\mathbf{d}$ by $\lambda(w \mid \mathbf{d})$, the probability that the person will gist $\mathbf{d}$ into $\mathbf{g} = \{g_1, g_2, \ldots g_n\}$ is

$$p(\mathbf{g} \mid \mathbf{d}) = \phi(n) \prod_{i=1}^{n} \lambda(g_i \mid \mathbf{d}).$$

Though this model is simplistic, it makes one plausible assumption: the more frequently a word appears in a document, the more likely it is to be included in a gist of that page. This algorithm is essentially identical (albeit in a different setting) to the language modelling approach to document retrieval introduced by Ponte and Croft [68], and also to Model 0, introduced in Section 3.3.1.

II. Accounting for unseen words

Algorithm 7 is limited in a number of ways, one of which is that the generated summaries can only contain words from the input document. A logical extension is to relax this restriction by allowing the gist to contain words not present in the source document. The idea is to draw (as before) a word according to the word frequencies in the input document, but then replace the drawn word with a related word—a synonym, perhaps, or a word used in similar contexts—before adding it to the gist.

Determining which word to substitute in place of the sampled word requires a probability distribution $\sigma(\cdot \mid w)$: if u is a very closely related word to v, then one would expect $\sigma(u \mid v)$ to be large. If the system recognizes W words, then the σ model is just a $W \times W$ stochastic matrix. (One could reasonably expect that the diagonal entries of this matrix, corresponding to "self-similarity" probabilities, will typically be large.) We will call this algorithm *expanded-lexicon gisting*, since the lexicon of candidate words for a summary of $\mathbf{d}$ are no longer just those appearing in $\mathbf{d}$.

This "draw then replace with a similar word" model of document gisting is similar to the IBM-style model of language translation [14]. The simplest of this family of statistical models pretends that a person renders a document into a different language by drawing words from it and translating each word—"draw, then translate" rather than "draw, then replace with a related word."

Algorithm 8: *Expanded-lexicon gisting*

Input: `Document` $\mathbf{d}$ `with word distribution` $\lambda(\cdot \mid \mathbf{d})$;
`Distribution` ϕ `over gist lengths;`
`Word-similarity model` $\sigma(\cdot \mid u)$ `for all words` w

Output: Gist $\mathbf{g}$ of $\mathbf{d}$

1. `Select a length for the gist:` $n \sim \phi$
2. `Do for` $i = 1$ `to` n
3. `Pick a word from the document:` $u \sim \lambda(\cdot \mid \mathbf{d})$
4. `Pick a replacement for that word:` $v \sim \sigma(\cdot \mid w)$
5. `Set` $\mathbf{g}_i = v$

As before, one can write down an expression for the probability that a person following this procedure will select, for an input document **d**, a specific gist $\mathbf{g} = \{g_1, g_2, \ldots g_n\}$. Assuming **d** contains m words,

$$p(\mathbf{g} \mid \mathbf{d}) = \phi(n) \prod_{i=1}^{n} p(g_i \mid \mathbf{d}) \tag{4.2}$$
$$= \phi(n) \prod_{i=1}^{n} \left(\frac{1}{m}\right) \sum_{j=1}^{m} \sigma(g \mid d_j)$$

In form, Algorithm 8 is a recast version of Model 1, described in Chapter 3, where the queries have now become summaries. However, the context in the two cases is quite different. In document ranking, the task is to assign a score $p(\mathbf{q} \mid \mathbf{d})$ to each of a set of documents $\{d_1, d_2 \ldots d_n\}$. In gisting, the task is to *construct* (synthesize) a gist g for which $p(\mathbf{g} \mid \mathbf{d})$ is highest. Secondly, the word-independence assumption which appears in $p(\mathbf{q} \mid \mathbf{d})$ and $p(\mathbf{g} \mid \mathbf{d})$ is relatively innocuous in document ranking, where word order in queries is often ignored at essentially no cost by IR systems. In gisting, however, word order is of potentially great importance: a fluent candidate summary is to be preferred over a disfluent one consisting of the same words, rearranged.

III. Generating readable summaries

One can extend Algorithm 8 by enforcing that the sequence of words comprising a candidate gist are coherent. For instance, one could ensure that two prepositions never appear next to each other in a gist. The next algorithm attempts to capture a notion of syntactic regularity by scoring candidate gists not only on how well they capture the essence (the process of content selection) of the original document, but also how coherent they are as a string of English words.

The coherence or readability of an n-word string $\mathbf{g} = \{g_1, g_2, \ldots g_n\}$ comprising a candidate gist is the *a priori* probability of seeing that string of words in text, which will appear as $p(\mathbf{g})$. One can factor $p(\mathbf{g})$ into a product of conditional probabilities as

$$p(\mathbf{g}) = \prod_{i=1}^{n} p(g_i \mid g_1, g_2 \ldots g_{i-1})$$

In practice, one can use a *trigram* model for $p(\mathbf{g})$, meaning that

$$p(g_i \mid g_1, g_2 \ldots g_{i-1}) \approx p(g_i \mid g_{i-2} g_{i-1}) \tag{4.3}$$

Although n-gram models of language make a quite strong (and clearly false) locality assumption about text, they have nonetheless proven successful in many human language technologies, including speech and optical character recognition [42, 63].

To devise a formal model of gisting which accounts for both readability and fidelity to the source document, we apply Bayes' Rule to (4.1):

$$\begin{aligned} \mathbf{g}^\star &= \arg\max_{\mathbf{g}} p(\mathbf{g} \mid \mathbf{d}) \\ &= \arg\max_{\mathbf{g}} p(\mathbf{d} \mid \mathbf{g})\; p(\mathbf{g}). \end{aligned} \tag{4.4}$$

According to (4.4), the optimal gist is the product of two terms: first, a fidelity term $p(\mathbf{d} \mid \mathbf{g})$, measuring how closely $\mathbf{d}$ and $\mathbf{g}$ match in content, and a readability term $p(\mathbf{g})$, measuring the *a priori* coherence of the gist $\mathbf{g}$.

For the readability term, one can use the language model (4.3). For the content proximity model $p(\mathbf{d} \mid \mathbf{g})$, one can simply reverse the direction of (4.2):

$$\begin{aligned} p(\mathbf{d} \mid \mathbf{g}) &= \hat{\phi}(m) \prod_{i=1}^{n} p(d_i \mid \mathbf{g}) \\ &= \hat{\phi}(m) \prod_{i=1}^{n} \sum_{j=1}^{m} \left(\frac{1}{n}\right) \sigma(d \mid g_j) \end{aligned} \tag{4.5}$$

Here $\hat{\phi}$ is a length distribution on *documents*, which system designers would in general wish to distinguish from the length distribution on summaries[1].

Algorithm 9: *Readable gisting*

Input: `Document` $\mathbf{d}$ `with word distribution` $\lambda(\cdot \mid \mathbf{d})$;

`Distribution` ϕ `over gist lengths;`

`Word-similarity model` $\sigma(\cdot \mid w)$ `for all words` w

`Trigram language model` $p(\mathbf{g})$ `for gists`

Output: Gist $\mathbf{g}$ of $\mathbf{d}$

1. `Select a length` n `for the gist:` $n \sim \phi$
2. `Search for the sequence g` $= \{g_1, g_2, \ldots g_n\}$ `maximizing` $p(\mathbf{d} \mid \mathbf{g})p(\mathbf{g})$

One can think of $p(\mathbf{g})$ as a prior distribution on candidate gists, and $p(\mathbf{d} \mid \mathbf{g})$ as the probability that the document $\mathbf{d}$ would arise from the gist $\mathbf{g}$.

One way to make sense of the seeming reverse order of prediction in (4.4) is with the source-channel framework from information theory. Imagine that the document to be gisted

[1] OCELOT's task is to find the best gist of a document, and the $\hat{\phi}$ term will contribute equally to every candidate gist. We can therefore ignore this term from now on.

was originally itself a gist: the germ of an idea in the imagination of whoever composed the page. The actual composition of the web page is like a corruption of this original message, and the goal of a gisting algorithm is to recover, from the web page itself, this original, hidden, ideal gist. In doing so, the algorithm makes use of a model $p(\mathbf{d} \mid \mathbf{g})$ of how ideas for web pages get converted ("corrupted") into pages, and a model $p(\mathbf{g})$ for what constitutes an *a priori* likely and unlikely gist. Figure 4.1 illustrates this information-theoretic interpretation.

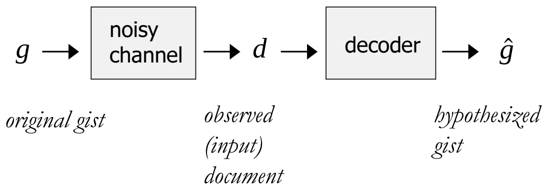


Figure 4.1: Gisting from a source-channel perspective

Algorithm 9 leaves unspecified the somewhat involved matter of searching for the optimal **g**. Speech and handwriting recognition systems face a similar problem in attempting to generate a transcription of a detected signal (an acoustic or written signal) which both accounts for the perceived signal and is a coherent string of words. As mentioned earlier, the most successful technique has been to apply a Viterbi-type search procedure, and this is the strategy that OCELOT adopts.

4.4 A source of summarized web pages

Applying machine learning to web-page gisting requires a large collection of gisted web pages for training. As mentioned previously, a suitable corpus for this task can be obtained from the Open Directory Project (`http://dmoz.org`). What makes Open Directory useful for learning to gist is that each of its entries—individual web sites—is summarized manually, by a human Open Directory volunteer.

For the experiments reported here, an automated script attempted to download each of the Open Directory's 868,227 web pages[2], along with the directory's description of each site. Since individual web sites oftem restrict requests for data from automatic programs ("spiders"), many of the pages were inaccessible. Those that were accessible were subject to the following processing:

[2]The directory is growing quickly, and at last count was approaching two million entries.

- Normalize text: remove punctuation, convert all text to lowercase; replace numbers by the symbol NUM; remove each occurrence of the 100 most common overall words (stopword-filtering). Though nothing in the algorithms requires the excision of stopwords, doing so yields a marked speedup in training.
- Remove all links, images, and meta-information
- Remove pages containing adult-oriented content[3];
- Remove HTML markup information from the pages;
- Remove pages containing frames;
- Remove pages that had been moved since their original inclusion in the Open Directory; in other words, pages containing just a "Page not found" message.
- Remove pages or gists that were too short—less than 400 or 60 characters, respectively. Pages that are too short are likely to be "pathological" in some way—often a error page delivered by the origin server indicating that a password is required to view the document, or the document has moved to a different location, or a certain type of browser is required to view the document.
- Remove duplicate web pages;
- Partition the remaining set of pairs into a training set (99%) and a test set (1%). (Traditionally when evaluating a machine learning algorithm, one reserves more than this fraction of the data for testing. But one percent of the Open Directory dataset comprises over a thousand web pages, which was sufficient for the evaluations reported below.)

At the conclusion of this process, 103,064 summaries and links remained in the training set, and 1046 remained in the test set. Figure 4.2 shows a "before and after" example of this filtering process on a single web page, along with Open Directory's summary of this page. After processing, the average length of the summaries was 13.6 words, and the average length of the documents was 211.1 words.

4.5 Training a statistical model for gisting

This section discusses the training of various statistical models for the specific task of gisting web pages.

[3] Skipping the pages listed in the `Adult` hierarchy goes far, but not the entire way, towards solving this problem.

***Filtered*:** `svenska sidan utsigten antik kuriosa welcome we sell and buy antiques and collectibles of good quality our shop is in central karlskrona sweden at borgmstarekajen close to the county museum and fisktorget see the map you will find swedish porcelain china glass and textiles here we are specialized in porcelain from karlskrona we have been in business since NUM welcome to our shop our opening hours are tuesday wednesday and thursday NUM NUM NUM NUM saturday NUM NUM NUM NUM other times on agreement bookmark this site copyright NUM utsigten antik kuriosa updated NUM NUM NUM contact us with email to utsigtenantikviteter net or phone NUM NUM`

Open Directory gist: `sell and buy antiques and collectibles of good quality our shop is in central karlskrona sweden`

Figure 4.2: A web page (top), after filtering (middle), and the Open Directory-provided gist of the page (bottom). Interestingly, the Open Directory gist of the document, despite being produced by a human, is rather subpar; it's essentially the first two sentences from the body of the web page.

4.5.1 Estimating a model of word relatedness

Recall that in Algorithm 9, the underlying statistical model $p(\mathbf{d} \mid \mathbf{g})$ which measures the "proximity" between a web page $\mathbf{d}$ and a candidate gist $\mathbf{g}$ is a generative model, predicting $\mathbf{d}$ from $\mathbf{g}$. This model factors, as seen in (4.5), into a product of sums of $\sigma(d \mid g)$ terms: the probability that a word g in a gist of a web page gives rise to a word d in the page itself. What follows is a description of how one can learn these word-to-word "relatedness" probabilities automatically from a collection of summarized web pages.

If there are W_g different recognized words in gists and W_p different recognized words in web pages, then calculating the parameters of the individual σ models is equivalent to filling in the entries of a $W_g \times W_p$ stochastic matrix. As mentioned above, there exist algorithms, first developed in the context of machine translation [14], for estimating maximum-likelihood values for the entries of this matrix using a collection of bilingual text. In this case, the two "languages" are the verbose language of documents and the succinct language of gists.

For the purposes of estimating the σ parameters, we re-introduce the notion of an alignment a between sequences of words, which in this case captures how words in gists produce the words in a web page. OCELOT also makes use of an artificial NULL added to position zero of every gist, whose purpose is to generate those words in the web page not strongly correlated with any other word in the gist.

Using a, $p(\mathbf{d} \mid \mathbf{g})$ decomposes in a by-now familiar way:

$$p(\mathbf{d} \mid \mathbf{g}) = \sum_a p(\mathbf{d}, a \mid \mathbf{g}) = \sum_a p(\mathbf{d} \mid a, \mathbf{g}) p(a \mid \mathbf{g}) \tag{4.6}$$

Making the simplifying assumption that to each word in $\mathbf{d}$ corresponds exactly one "parent" word in $\mathbf{g}$ (possibly the null word), one can write

$$p(\mathbf{d} \mid a, \mathbf{g}) = \prod_{i=1}^{m} \sigma(d_i \mid g_{a_i}) \tag{4.7}$$

Here g_{a_i} is the gist word aligned with the ith web page word. Figure 4.3 illustrates a sample alignment between a small web page and its summary.

If $\mathbf{d}$ contains m words and $\mathbf{g}$ contains $n+1$ words (including the null word), there are $(n+1)^m$ alignments between $\mathbf{g}$ and $\mathbf{d}$. By assuming that all these alignments are equally likely allows us to write

$$p(\mathbf{d} \mid \mathbf{g}) = \frac{p(m \mid \mathbf{g})}{(n+1)^m} \sum_A \prod_{i=1}^{m} \sigma(d_i \mid g_{a_i}) \tag{4.8}$$

OCELOT views the Open Directory dataset as a collection of web pages and their summaries, $\mathcal{C} = \{(\mathbf{d}_1, \mathbf{g}_1), (\mathbf{d}_2, \mathbf{g}_2), (\mathbf{d}_3, \mathbf{g}_3) \ldots$. The likelihood method suggests that one should

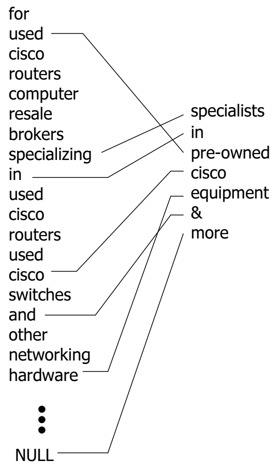


Figure 4.3: One of the exponentially many alignments between this imaginary document/gist pair. Calculating the score $p(\mathbf{d} \mid \mathbf{g})$ of a document/gist pair involves, implicitly, a sum over *all* possible ways of aligning the words. This diagram is analogous to Figure 3.3, though now the righthand column corresponds to a summary, rather than a query.

adjust the parameters of (4.8) in such a way that the model assigns as high a probability as possible to $\mathcal{C}$. This maximization must be performed subject to the constraints $\sum_d \sigma(d \mid g) = 1$ for all words g. Using Lagrange multipliers,

$$\sigma(d \mid g) = Z \sum_a p(\mathbf{d}, a \mid \mathbf{g}) \sum_{j=1}^{m} \delta(d, d_j)\delta(g, g_{a_j}), \tag{4.9}$$

where Z is a normalizing factor and δ is the Kronecker delta function.

The parameter $\sigma(d \mid g)$ appears explicitly in the left-hand side of (4.9), and implicitly in the right. By repeatedly solving this equation for all pairs d, g (in other words, applying the EM algorithm), one eventually reaches a stationary point of the likelihood.

Equation (4.9) contains a sum over alignments, which is exponential and suggests that the computing the parameters in this way is infeasible. In fact, just as with (3.11), we can rewrite the expression in a way that leads to a more efficient calculation:

$$\sum_a \prod_{i=1}^{m} \sigma(d_i \mid g_{a_i}) = \prod_{i=1}^{m} \sum_{j=0}^{n} \sigma(d_i \mid g_j) \tag{4.10}$$

This rearranging means that computing $\sum_a p(\mathbf{d}, a \mid \mathbf{g})$ requires only $\Theta(mn)$ work, rather than $\Theta(n^m)$.

Figure 4.4 shows the progress of the perplexity of the Open Directory training data during the six iterations of training, using all the 103,064 gist/web page pairs in the training set (24,231,164 words in the web page data and 1,922,393 words in the summaries). The vocabularies were constructed from the top 65535 words appearing at least twice; all other words were mapped to the symbol OOV (for "out of vocabulary").

Table 4.1 shows the top entries for a few selected words.

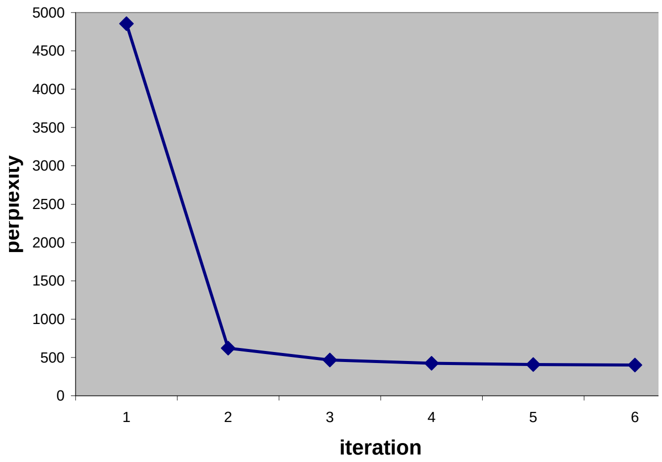


Figure 4.4: Decrease in perplexity of the training set during the six iterations of the EM algorithm

4.5.2 Estimating a language model

OCELOT attempts to ensure that its hypothesized gists are readable with the help of a trigram model of the form (4.3). For a W-word vocabulary, such a model is characterized by W^3 parameters: $p(w \mid u, v)$ is the probability that the word w follows the bigram u, v.

Constructing such a model involved calculating $p(w \mid u, v)$ values from the full training set of Open Directory gists. Building the language model consisted of the following steps:

1. Construct a vocabulary of active words from those words appearing at least twice within the collection of summaries. This amounted to 37,863 unique words.

2. Build a trigram word model from this data using maximum-likelihood estimation.

3. "Smooth" this model (by assigning some probability mass to unseen trigrams) using the Good-Turing estimate [35].

To accomplish the final two steps, OCELOT uses the publicly-available CMU-Cambridge Language Modelling Toolkit [21].

`job`	`job` 0.194	`jobs` 0.098	`career` 0.028	`employment` 0.028
`wilderness`	`wilderness` 0.123	`the` 0.061	`national` 0.032	`forest` 0.028
`associations`	`associations` 0.083	`association` 0.063	OOV 0.020	`members` 0.013
`ibm`	`ibm` 0.130	`business` 0.035	`solutions` 0.019	`support` 0.017
`camera`	`camera` 0.137	`cameras` 0.045	`photo` 0.020	`photography` 0.014
`investments`	`investments` 0.049	`investment` 0.046	`fund` 0.033	`financial` 0.025
`contractor`	`contractor` 0.080	`contractors` 0.030	`construction` 0.027	`our` 0.016
`quilts`	`quilts` 0.141	`quilt` 0.074	`i` 0.036	`quilting` 0.034
`exhibitions`	`exhibitions` 0.059	OOV 0.056	`art` 0.048	`museum` 0.041
`ranches`	`ranches` 0.089	`springs` 0.034	`colorado` 0.032	`ranch` 0.030

Table 4.1: Word-relatedness models $\sigma(\cdot \mid w)$ for selected words w, computed in an unsupervised manner from the Open Directory training data.

4.6 Evaluation

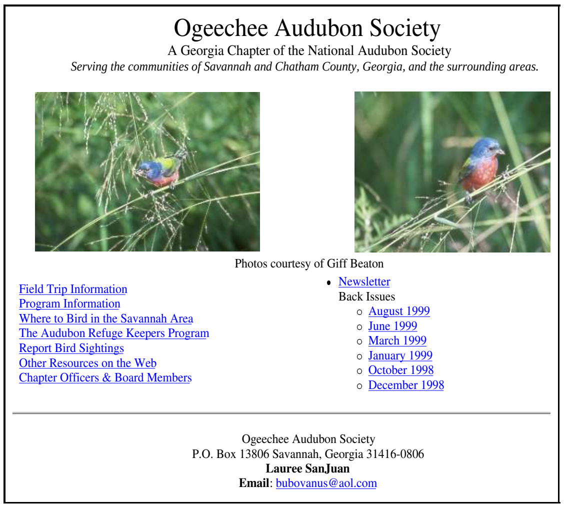
Summarization research has grappled for years with the issue of how to perform a rigorous evaluation of a summarization system [34, 38, 44, 71]. One can categorize summarization evaluations as

- *extrinsic*: evaluating the summary with respect to how useful it is when embedded in some application;
- *intrinsic*: adjuticating the merits of the summary on its own terms, without regard to its intended purpose.

This section reports on one extrinsic and two intrinsic evaluations, using Algorithm 9.

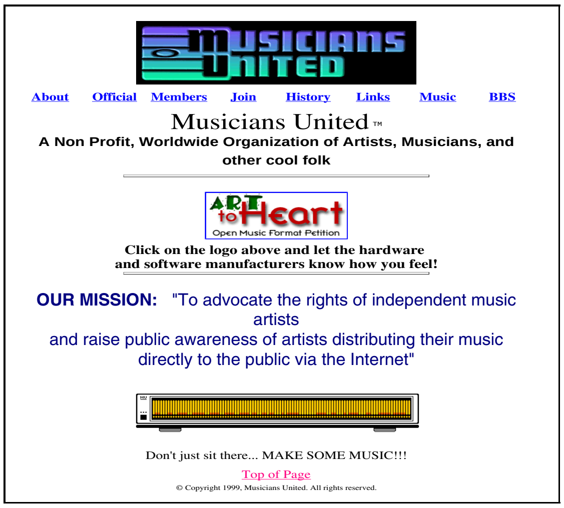
4.6.1 Intrinsic: evaluating the language model

Since OCELOT uses both a language model and a word-relatedness model to calculate a gist of a web page, isolating the contribution of the language model to the performance of OCELOT is a difficult task. But the speech recognition literature suggest a strategy: gauge the performance of a language model in isolation from the rest of the summarizer by measuring how well it predicts a previously-unseen collection $\mathcal{G}$ of actual summaries. Specifically, one can calculate the probability which the language model assigns to a set of unseen Open Directory gists; the higher the probability, the better the model.



Open Directory gist: `a chapter of the national audubon society serving the communities of savannah chatham county and the surrounding areas`

OCELOT *gist:* `audubon society atlanta area savannah georgia chatham and local birding savannah keepers chapter of the audubon georgia and leasing`



Open Directory gist: `to advocate the rights of independent music artists and raise public awareness of artists distributing their music directly to the public via the internet`

OCELOT *gist:* `the music business and industry artists raise awareness rock and jazz`

Figure 4.5: Selected output from OCELOT. The original web page is shown above with the actual and hypothesized gists below.

The log-likelihood assigned by λ to an n-word collection $\mathcal{G}$ is

$$\log p(\mathcal{G}) = \sum_{i=1}^{n} \log p(g_i \mid g_{i-2} g_{i-1})$$

As described in Chapter 2, the *perplexity* of $\mathcal{G}$ according to the trigram model is related to $\log p(\mathcal{G})$ by

$$\Pi(\mathcal{G}) = \exp\left\{ - \left(\frac{1}{n}\right) \sum_{i=1}^{n} \log p(g_i \mid g_{i-2} g_{i-1}) \right\}$$

Roughly speaking, perplexity can be thought of as the average number of "guesses" the language model must make to identify the next word in a string of text comprising a gist drawn from the test data. An upper bound in this setting is $\mid W \mid = 37,863$: the number of different words which could appear in any single position in a gist. To the test collection of 1046 gists consisting of 20,775 words, the language model assigned a perplexity of 362.

This is to be compared with the perplexity of the same text as measured by the weaker bigram and unigram models: 536 and 2185, respectively. The message here is that, at least in an information theoretic sense, using a trigram model to enforce fluency on the generated summary is superior to using a bigram or unigram model (the latter is what is used in "bag of words" gisting).

4.6.2 Intrinsic: gisted web pages

Figure 4.5 shows two examples of the behavior of OCELOT on web pages selected from the evaluation set of Open Directory pages—web pages, in other words, which OCELOT did not observe during the learning process.

The generated summaries do leave something to be desired. While they capture the essence of the source document, they are not very fluent. The performance of the system could clearly benefit from more sophisticated content selection and surface realization models. For instance, even though Algorithm 9 strives to produce well-formed summaries with the help of a trigram model of language, the model makes no effort to preserve word order between document and summary. OCELOT has no mechanism, for example, for distinguishing between the documents `Dog bites man` and `Man bites dog`. A goal for future work is to consider somewhat more sophisticated stochastic models of language, investigating more complex approaches such as longer range Markov models, or even more structured syntactic models, such as the ones proposed by Chelba and Jelinek [19, 20]. Another possibility is to consider a hybrid extractive/non-extractive system: a summarizer which builds a gist form entire phrases from the source document where possible, rather than just words.

4.6.3 Extrinsic: text categorization

For an extrinsic evaluation of automatic summarization, we developed a user study to assess how well the automatically-generated summary of a document helps a user classify a document into one of a fixed number of categories.

Specifically, we collected a set of 629 web pages along with their human-generated summaries, made available by the OpenDirectory project. The pages were roughly equally distributed across the following categories:

SPORTS/MARTIAL ARTS	SOCIETY/PHILOSOPHY
SPORTS/MOTORSPORTS	SOCIETY/MILITARY
SPORTS/EQUESTRIAN	HOME/GARDENS

For each page, we generated six different "views":

1. the text of the web page
2. the title of the page
3. an automatically-generated summary of the page
4. the OpenDirectory-provided, human-authored summary of the page
5. a set of words, equal in size to the automatically-generated summary, selected uniformly at random from the words in the original page
6. the leading sequence of words in the page, equal in length to the automatically-generated summary.

Taking an information theoretic perspective, one could imagine each of these views as the result of passing the original document through a different noisy channel. For instance, the title view results from passing the original document through a filter which distills a document into its title. With this perspective, the question addressed in this user study is this: how much information is lost through each of these filters? A better representation of a document, of course, loses less information.

To assess the information quality of a view, we ask a user to try to guess the proper (OpenDirectory-assigned) classification of the original page, using only the information in that view. For concreteness, Table 4.6.3 displays a single entry from among those collected for this study. Only very lightweight text normalization was performed: lowercasing all words, removing filenames and urls, and mapping numbers containing two or more digits to the canonical token `[num]`.

Table 4.6.3 contains the results of the user study. The results were collected from six different participants, each of whom classified approximately 120 views. The records assigned to each user were selected uniformly at random from the full collection of records, and the view for each record was randomly selected from among the six possible views.

Perhaps the most intriguing aspect of these results is how the human-provided summary was actually *more* useful to the users, on average, than the full web page. This isn't too surprising; after all, the human-authored summary was designed to distill the essence of the original page, which often contains extraneous information of tangential relevance. The synthesized summary performed approximately as well as the title of the page, placing it significantly higher than a randomly-generated summary but also inferior to both the original page and the human-generated summary.

full page: `davidcoulthard com david coulthard driving the [num] mclaren mercedes benz formula 1 racing car click here to enter copyright [num] [num] davidcoulthard com all rights reserved privacy policy davidcoulthard com is not affiliated with david coulthard or mclaren`

title: `david coulthard`

OpenDirectory human summary: `website on mclaren david coulthard the scottish formula 1 racing driver includes a biography racing history photos quotes a message board`

automatically-generated summary: `criminal driving teams automobiles formula 1 racing more issues shift af railings crash teams`

Randomly-selected words: `davidcoulthard policy click david mclaren driving to [num] com reserved with [num] copyright or`

Leading words: `davidcoulthard com david coulthard driving the [num] mclaren mercedes benz formula 1 racing car`

Table 4.2: A single record from the user study, containing the six different views of a single web page. This document was from the topic SPORTS/MOTORSPORTS.

4.7 Translingual gisting

With essentially no adaptation, OCELOT could serve as a translingual summarization system: a system for producing the gist of a document in another language. The only necessary ingredient is a collection of documents in one language with summaries in another: the word-relatedness matrix would then automatically become a matrix of translation probabilities. Experts in the field of information retrieval consider translingual summarization to be a key ingedient in enabling universal access to electronic information—-in other words, the internationalization of the Internet [65].

Some initial proof-of-concept experiments to generate English summaries of French web pages suggest that OCELOT may indeed be useful in this setting. For this purpose, one can use the same language model on (English) summaries as in Section 4.6. Since locating a suitably large parallel corpus of French web pages and English summaries from which to estimate σ is difficult, we were forced to use a pre-built translation model, constructed from the proceedings of the Canadian parliament—the Hansards described in Chapter 1.

view	# of samples	# correct	accuracy
OpenDirectory human-provided summary	109	94	0.862
Words in original page	131	108	0.824
First n words in page	98	75	0.765
Title of page	112	80	0.714
Synthesized summary	115	80	0.695
Words randomly-selected from page	122	76	0.622

Table 4.3: Results of an extrinsic user study to assess the quality of the automatically-generated web page summaries.

The subset of the Hansard corpus used to estimate this model contained two million parallel English/French sentences, comprising approximately 43 million words. Using a model trained on parliamentary discourse on the domain of web page gisting has its shortcomings: words may sometimes have quite different statistics in the Hansards than in the average web page. One potential line of future work involves using a web spidering tool to identify and download web pages published in different languages [73].

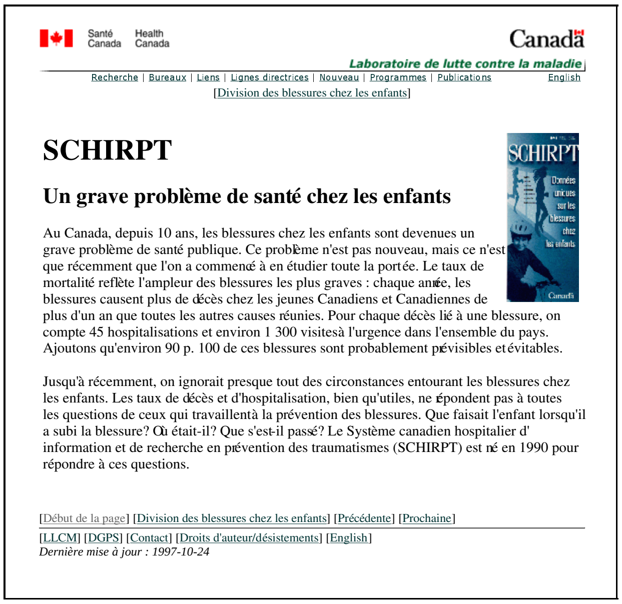
Figure 4.6 gives an example of French web page gisted into English.

4.8 Chapter summary

This chapter has described the philosophy, architecture, and performance of OCELOT, a prototype web-page summarization system. OCELOT is designed to generate non-extractive summaries of a source document; in fact, the generated summaries are likely to contain words not even appearing in the original document. This approach to summarization appears particularly well-suited to web pages, which are often disjointed lists of phrases and links not amenable to traditional extraction-based techniques.

As it stands, OCELOT represents but an initial foray into automating the process of web page gisting. Considerably more sophisticated models will be required to produce useful, readable summaries. As mentioned, considerably more effort will have to go into evaluation—via user studies, most probably—in order to assess the relative extrinsic quality of competing summarization techniques.

Asked to summarize a web page, a reasonably intelligent person would no doubt make use of information that OCELOT ignores. For instance, text often appears in web pages in the form of bitmapped images, but this information is lost without a front-end OCR module to extract this text. Also, the system does not exploit structural clues about what's



Santé Canada Health Canada

Canada

Laboratoire de lutte contre la maladie

Recherche | Bureaux | Liens | Lignes directrices | Nouveau | Programmes | Publications English

[Division des blessures chez les enfants]

SCHIRPT

Un grave problème de santé chez les enfants

Au Canada, depuis 10 ans, les blessures chez les enfants sont devenues un grave problème de santé publique. Ce problème n'est pas nouveau, mais ce n'est que récemment que l'on a commencé à en étudier toute la portée. Le taux de mortalité reflète l'ampleur des blessures les plus graves : chaque année, les blessures causent plus de décès chez les jeunes Canadiens et Canadiennes de plus d'un an que toutes les autres causes réunies. Pour chaque décès lié à une blessure, on compte 45 hospitalisations et environ 1 300 visitesà l'urgence dans l'ensemble du pays. Ajoutons qu'environ 90 p. 100 de ces blessures sont probablement prévisibles etévitables.

Jusqu'à récemment, on ignorait presque tout des circonstances entourant les blessures chez les enfants. Les taux de décès et d'hospitalisation, bien qu'utiles, ne répondent pas à toutes les questions de ceux qui travaillentà la prévention des blessures. Que faisait l'enfant lorsqu'il a subi la blessure? Où était-il? Que s'est-il passé? Le Système canadien hospitalier d' information et de recherche en prévention des traumatismes (SCHIRPT) est né en 1990 pour répondre à ces questions.

[Début de la page] [Division des blessures chez les enfants] [Précédente] [Prochaine]

[LLCM] [DGPS] [Contact] [Droits d'auteur/désistements] [English]

Dernière mise à jour : 1997-10-24

OCELOT ***gist:*** `health protection branch of the protection of health anti inflation guidelines health of animals in volume may table of contents of our children in central canada review of u.s beginning at page volume final vote day may`

Figure 4.6: Selected output from a French-English version of OCELOT

important on the page. For instance, the text within the `<title>` ... `</title>` region is likely to be relatively important, while text within a `<small>` ... `</small>` is probably less important.

Algorithm 9 is too resource-intensive to be a real-time procedure. In fact, on the workstation used in the experiments here, calculating a gist using this algorithm could take as long as a few minutes. An important next step is to devise an efficient, approximate version of this algorithm, in the spirit of the FastRank procedure described in the previous chapter.

Another next step, as mentioned earlier, is to consider using phrases extracted from the source document as atomic units—just like individual words—in piecing together a summary. That is, the candidate constituents of a summary of a document are all the words known to the system, plus a set of phrases appearing in the source document. Perhaps this strategy is a step back towards extractive summarization, but considering phrases from the source document might increase the chance for a fluent summary.

Chapter 5

Query-relevant summarization

This chapter addresses the problem of query-relevant summarization: succinctly characterizing the relevance of a document to a query. Learning parameter values for the proposed statistical models requires a large collection of summarized documents, which is difficult to obtain. In its place, we propose the use of a collection of FAQ (frequently-asked question) documents. Taking a learning approach enables a principled, quantitative evaluation of the proposed system, and the results of some initial experiments—on a collection of Usenet FAQs and on a FAQ-like set of customer-submitted questions to several large retail companies—suggest the plausibility of learning for summarization.

5.1 Introduction

An important distinction in document summarization is between *generic summaries*, which capture the central ideas of the document in much the same way that the abstract above was designed to distill this chapter's salient points, and *query-relevant summaries*, which reflect the relevance of a document to a user-specified query. Chapter 4 described a method for generating generic summaries of a document, while the focus here is on query-relevant summarization, sometimes called "user-focused" summarization [55].

Query-relevant summaries are especially important in the "needle(s) in a haystack" document retrieval problem tackled in Chapter 3: a user has an information need expressed as a query (`''What countries export smoked salmon?''` or maybe just `''export smoked salmon''`), and a retrieval system must locate within a large collection of documents those documents most likely to fulfill this need. Many interactive retrieval systems—commercial web search engines, for instance—present the user with a small set of candidate relevant documents, each summarized; the user must then perform a kind of triage to identify likely relevant documents from this set. The web page summaries presented by most search engines are generic, not query-relevant, and thus provide very little guidance to the user in assessing relevance. This reliance on query-independent summaries is attributable in part to economic reasons: summarizing a document *de novo* for every query is more expensive than summarizing the document once and for all, independent of any specific query. But query-relevant summarization (QRS) can provide a more effective characterization of a document by accounting for the user's information need when generating a summary.

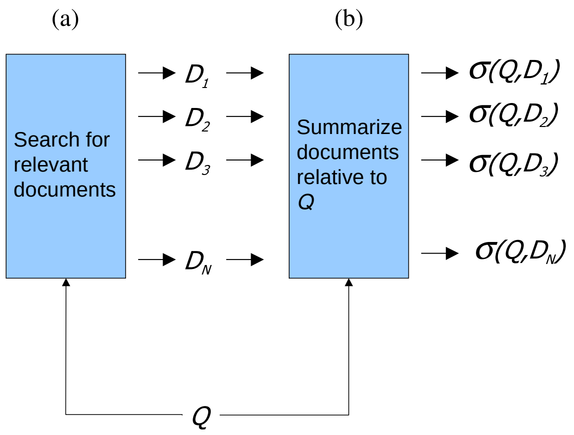


Figure 5.1: One promising setting for query-relevant summarization is large-scale document retrieval. Starting from a user-specified query $\mathbf{q}$, search engines typically first (a) identify a set of documents which appear potentially relevant to the query, and then (b) produce a short characterization $\sigma(\mathbf{d}, \mathbf{q})$ of each document's relevance to $\mathbf{q}$. The purpose of $\sigma(\mathbf{d}, \mathbf{q})$ is to help the user decide which of the documents merits a more detailed inspection.

As with almost all previous work on summarization (excluding the previous chapter, of course), this chapter focuses on the task of *extractive summarization*: selecting as summaries text spans—either complete sentences or paragraphs—from the original document.

5.1.1 Statistical models for summarization

From a document $\mathbf{d}$ and query $\mathbf{q}$, the task of query-relevant summarization is to extract a portion $\mathbf{s}$ from $\mathbf{d}$ which best reveals how the document relates to the query. To begin, we start with a collection $\mathcal{C}$ of $\{\mathbf{d}, \mathbf{q}, \mathbf{s}\}$ triplets, where $\mathbf{s}$ is a human-constructed summary of $\mathbf{d}$ relative to the query $\mathbf{q}$. From such a collection of data, we fit the best function $\sigma : (\mathbf{q}, \mathbf{d}) \rightarrow \mathbf{s}$ mapping document/query pairs to summaries.

The mapping used here is a probabilistic one, meaning the system assigns a value

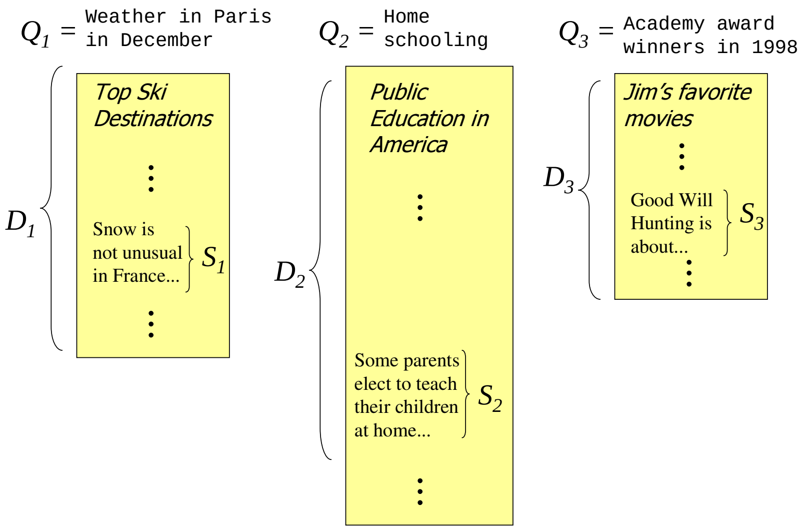


Figure 5.2: Learning to perform query-relevant summarization requires a set of documents summarized with respect to queries. The diagram shows three imaginary triplets $\{\mathbf{d}, \mathbf{q}, \mathbf{s}\}$, though the statistical learning techniques described in Section 5.2 require many thousands of examples.

$p(\mathbf{s} \mid \mathbf{d}, \mathbf{q})$ to each candidate summary $\mathbf{s}$ of $(\mathbf{d}, \mathbf{q})$. The QRS system will summarize a $(\mathbf{d}, \mathbf{q})$ pair by selecting

$$\sigma(\mathbf{d}, \mathbf{q}) \stackrel{\text{def}}{=} \arg\max_{\mathbf{s}} p(\mathbf{s} \mid \mathbf{d}, \mathbf{q})$$

There are at least two ways to interpret $p(\mathbf{s} \mid \mathbf{d}, \mathbf{q})$. First, one could view $p(\mathbf{s} \mid \mathbf{d}, \mathbf{q})$ as a "degree of belief" that the correct summary of $\mathbf{d}$ relative to $\mathbf{q}$ is $\mathbf{s}$. Of course, what constitutes a good summary in any setting is subjective: any two people performing the same summarization task will likely disagree on which part of the document to extract. One could, in principle, ask a large number of people to perform the same task. Doing so would impose a distribution $p(\cdot \mid \mathbf{d}, \mathbf{q})$ over candidate summaries. Under the second, or "frequentist" interpretation, $p(\mathbf{s} \mid \mathbf{d}, \mathbf{q})$ is the fraction of people who would select $\mathbf{s}$—equivalently, the probability that a person selected at random would prefer $\mathbf{s}$ as the summary. (This frequentist interpretation is similar to the interpretation of $p(\mathbf{g} \mid \mathbf{d})$ in Section 4.3.)

The statistical model $p(\cdot \mid \mathbf{d}, \mathbf{q})$ is parametric, the values of which are learned by inspection of the $\{\mathbf{d}, \mathbf{q}, \mathbf{s}\}$ triplets. The learning process involves maximum-likelihood estimation of probabilistic language models and the statistical technique of shrinkage [81].

This probabilistic approach easily generalizes to the generic summarization setting, where there is no query. In that case, the training data consists of $\{\mathbf{d}, \mathbf{s}\}$ pairs, where $\mathbf{s}$ is a summary of the document $\mathbf{d}$. The goal, in this case, is to learn and apply a mapping

$\tau : \mathbf{d} \to \mathbf{s}$ from documents to summaries. That is, find

$$\tau(\mathbf{d}) \stackrel{\text{def}}{=} \arg\max_{\mathbf{s}} p(\mathbf{s} \mid \mathbf{d})$$

5.1.2 Using FAQ data for summarization

This chapter has proposed using statistical learning to construct a summarization system, but has not yet discussed the one crucial ingredient of any learning procedure: training data. The ideal training data would contain a large number of heterogeneous documents, a large number of queries, and summaries of each document relative to each query. We know of no such publicly-available collections. Many studies on text summarization have focused on the task of summarizing newswire text, but there is no obvious way to use news articles for query-relevant summarization within the framework proposed here.

This chapter proposes a novel data collection for training a QRS model: frequently-asked question documents. Each frequently-asked question document (FAQ) is comprised of questions and answers about a specific topic. One can view each answer in a FAQ as a summary of the document relative to the question which preceded it. That is, an FAQ with N question/answer pairs comes equipped with N different queries and summaries: the answer to the kth question is a summary of the document relative to the kth question. While a somewhat unorthodox perspective, this insight allows us to enlist FAQs as labeled training data for the purpose of learning the parameters of a statistical QRS model. (Sato and Sato also used FAQs as a source of summarization corpora, but their approach was quite different from that presented here, and did not use either statistical models or machine learning [77].)

FAQ data has some properties that make it particularly attractive for text learning:

- There exist a large number of Usenet FAQs—several thousand documents—publicly available on the Web[1]. Moreover, many large companies maintain their own FAQs to streamline the customer-response process.
- FAQs are generally well-structured documents, so the task of extracting the constituent parts (queries and answers) is amenable to automation. There have even been proposals for standardized FAQ formats, such as RFC1153 and the Minimal Digest Format [85].
- Usenet FAQs cover an astonishingly wide variety of topics, ranging from extraterrestrial visitors to mutual-fund investing. If there's an online community of people with a common interest, there's likely to be a Usenet FAQ on that subject.

[1]Two online sources for FAQ data are `www.faqs.org` and `rtfm.mit.edu`.

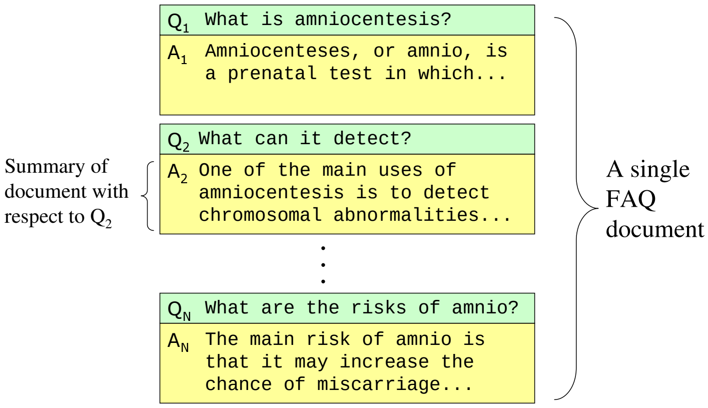


Figure 5.3: FAQs consist of a list of questions and answers on a single topic; the FAQ depicted here is part of an informational document on amniocentesis. This chapter views answers in a FAQ as different summaries of the FAQ: the answer to the kth question is a summary of the FAQ relative to that question.

5.2 A probabilistic model of summarization

Given a query $\mathbf{q}$ and document $\mathbf{d}$, the query-relevant summarization task is to find

$$\mathbf{s}^\star \equiv \arg\max_{\mathbf{s}} p(\mathbf{s} \mid \mathbf{d}, \mathbf{q}),$$

the *a posteriori* most probable summary for $(\mathbf{d}, \mathbf{q})$. Using Bayes' rule, one can rewrite this expression as

$$\begin{aligned} \mathbf{s}^\star &= \arg\max_{\mathbf{s}} p(\mathbf{q} \mid \mathbf{s}, \mathbf{d})\, p(\mathbf{s} \mid \mathbf{d}), \\ &\approx \arg\max_{\mathbf{s}} \underbrace{p(\mathbf{q} \mid \mathbf{s})}_{relevance}\ \underbrace{p(\mathbf{s} \mid \mathbf{d})}_{fidelity}, \end{aligned} \tag{5.1}$$

where the last line follows by dropping the dependence on $\mathbf{d}$ in $p(\mathbf{q} \mid \mathbf{s}, \mathbf{d})$.

Equation (5.1) is a search problem: find the summary $\mathbf{s}^\star$ which maximizes the product of two factors:

1. The **relevance** $p(\mathbf{q} \mid \mathbf{s})$ of the query to the summary: A document may contain some portions directly relevant to the query, and other sections bearing little or no relation to the query. Consider, for instance, the problem of summarizing a survey on the history of organized sports relative to the query "*Who was Lou Gehrig?*" A summary mentioning Lou Gehrig is probably more relevant to this query than one describing the rules of volleyball, even if two-thirds of the survey happens to be about volleyball.

2. The **fidelity** $p(\mathbf{s} \mid \mathbf{d})$ of the summary to the document: Among a set of candidate summaries whose relevance scores are comparable, we should prefer that summary $\mathbf{s}$ which is most representative of the document as a whole. Summaries of documents relative to a query can often mislead a reader into overestimating the relevance of an unrelated document. In particular, very long documents are likely (by sheer luck) to contain some portion which appears related to the query. A document having nothing to do with Lou Gehrig may include a mention of his name in passing, perhaps in the context of amyotropic lateral sclerosis, the disease from which he suffered. The fidelity term guards against this occurrence by rewarding or penalizing candidate summaries, depending on whether they are germane to the main theme of the document.

   More generally, the fidelity term represents a *prior*, query-independent distribution over candidate summaries. In addition to enforcing fidelity, this term could serve to distinguish between more and less fluent candidate summaries, in much the same way (as the previous chapter described) the trigram language model steers OCELOT towards a more fluent summary.

In words, (5.1) says that the best summary of a document relative to a query is relevant to the query (exhibits a large $p(\mathbf{q} \mid \mathbf{s})$ value) and also representative of the document from which it was extracted (exhibits a large $p(\mathbf{s} \mid \mathbf{d})$ value). What follows is a description of the parametric form of these models, and how to determine optimal values for these parameters using maximum-likelihood estimation.

5.2.1 Language modeling

One reasonable statistical model for both $p(\mathbf{q} \mid \mathbf{s})$ and $p(\mathbf{s} \mid \mathbf{d})$ is a unigram probability distribution over words; in other words, a language model.

The fidelity model $p(\mathbf{s} \mid \mathbf{d})$

One simple statistical characterization of an n-word document $\mathbf{d} = \{d_1, d_2, \ldots d_n\}$ is the frequency of each word in $\mathbf{d}$—in other words, a marginal distribution over words. That is, if word w appears k times in $\mathbf{d}$, then $p_{\mathbf{d}}(w) = k/n$. This is not only intuitive, but also the maximum-likelihood estimate for $p_{\mathbf{d}}(w)$.

Now imagine that, when asked to summarize $\mathbf{d}$ relative to $\mathbf{q}$, a person generates a summary from $\mathbf{d}$ in the following way:

1. Select a length m for the summary according to some distribution $l_{\mathbf{d}}$.
2. Do for $i = 1, 2, \ldots m$:
   - Select a word w at random according to the distribution $p_{\mathbf{d}}$. (That is, throw all the words in $\mathbf{d}$ into a bag, pull one out, and then replace it.)
3. Set $\mathbf{s}_i \leftarrow w$

In following this procedure, the person will generate the summary $\mathbf{s} = \{s_1, s_2, \ldots s_m\}$ with probability

$$p(\mathbf{s} \mid \mathbf{d}) = l_{\mathbf{d}}(m) \prod_{i=1}^{m} p_{\mathbf{d}}(s_i) \tag{5.2}$$

Denoting by $\mathcal{W}$ the set of all known words, and by $c(w \in \mathbf{d})$ the number of times that word w appears in $\mathbf{d}$, one can also write (5.2) as a multinomial distribution:

$$p(\mathbf{s} \mid \mathbf{d}) = l_{\mathbf{d}}(m) \prod_{w \,\in\, \mathcal{W}} p(w)^{c(w \,\in\, \mathbf{d})}. \tag{5.3}$$

This characterization of $\mathbf{d}$ amounts to a bag of words model, since the distribution $p_{\mathbf{d}}$ does not take account of the order of the words within the document $\mathbf{d}$, but rather views $\mathbf{d}$ as an unordered set. Of course, ignoring word order (an approximation which should be familiar to the reader by now) amounts to discarding potentially valuable information. In Figure 5.3, for instance, the second question contains an anaphoric reference to the preceding question: a sophisticated context-sensitive model of language might be able to detect that `it` in this context refers to `amniocentesis`, but a context-free model will not.

The relevance model $p(\mathbf{q} \mid \mathbf{s})$

In principle, one could proceed analogously to (5.2), and take

$$p(\mathbf{q} \mid \mathbf{s}) = l_{\mathbf{s}}(k) \prod_{i=1}^{m} p_{\mathbf{s}}(q_i). \tag{5.4}$$

for a length-k query $\mathbf{q} = \{q_1, q_2 \ldots q_k\}$. But this strategy suffers from a sparse estimation problem. In contrast to a document, which will typically contain a few hundred words, a normal-sized summary contains just a handful of words. What this means is that $p_{\mathbf{s}}$ will assign zero probability to most words, and any query containing a word not in the summary will receive a relevance score of zero.

(The fidelity model doesn't suffer from zero-probabilities, at least not in the extractive summarization setting. Since a summary $\mathbf{s}$ is part of its containing document $\mathbf{d}$, every word in $\mathbf{s}$ also appears in $\mathbf{d}$, and therefore $p_{\mathbf{d}}(s) > 0$ for every word $s \in \mathbf{s}$. But we have no guarantee, for the relevance model, that a summary contains all the words in the query.)

One way to address this zero-probability problem is by interpolating or "smoothing" the $p_{\mathbf{s}}$ model with four more robustly estimated unigram word models. Listed in order of decreasing variance but increasing bias away from $p_{\mathbf{s}}$, they are:

$p_{\mathcal{N}}$: a probability distribution constructed using not only $\mathbf{s}$, but also all words within the six summaries (answers) surrounding $\mathbf{s}$ in $\mathbf{d}$. Since $p_{\mathcal{N}}$ is calculated using more text than just $\mathbf{s}$ alone, its parameter estimates should be more robust that those of $p_{\mathbf{s}}$. On the other hand, the $p_{\mathcal{N}}$ model is, by construction, biased away from $p_{\mathbf{s}}$, and therefore provides only indirect evidence for the relation between $\mathbf{q}$ and $\mathbf{s}$.

$p_{\mathbf{d}}$: a probability distribution constructed over the entire document $\mathbf{d}$ containing $\mathbf{s}$. This model has even less variance than $p_{\mathcal{N}}$, but is even more biased away from $p_{\mathbf{s}}$.

$p_{\mathcal{C}}$: a probability distribution constructed over all documents $\mathbf{d}$.

$p_{\mathcal{U}}$: the uniform distribution over all words.

Figure 5.4 is a hierarchical depiction of the various language models which come into play in calculating $p(\mathbf{q} \mid \mathbf{s})$. Each summary model $p_{\mathbf{s}}$ lives at a leaf node, and the relevance $p(\mathbf{q} \mid \mathbf{s})$ of a query to that summary is a convex combination of the distributions at each node along a path from the leaf to the root[2]:

$$p(\mathbf{q} \mid \mathbf{s}) = \lambda_{\mathbf{s}} p_{\mathbf{s}}(\mathbf{q}) + \lambda_{\mathcal{N}} p_{\mathcal{N}}(\mathbf{q}) + \lambda_{\mathbf{d}} p_{\mathbf{d}}(\mathbf{q}) + \lambda_{\mathcal{C}} p_{\mathcal{C}}(\mathbf{q}) + \lambda_{\mathcal{U}} p_{\mathcal{U}}(\mathbf{q}) \tag{5.5}$$

Calculating the weighting coefficients $\lambda = \{\lambda_{\mathbf{s}}, \lambda_{\mathcal{N}}, \lambda_{\mathbf{d}}, \lambda_{\mathcal{C}}, \lambda_{\mathcal{U}}\}$ is a fairly straightforward matter using the statistical technique known as *shrinkage* [81], a simple form of the EM algorithm. Intuitively, the goal of this algorithm in this context is to calculate the relative "reliability" (predictive accuracy) of each of the constituent models, and assign a weight to each model in accord with its reliability.

As a practical matter, assuming the $l_{\mathbf{s}}$ model assigns probabilities independently of $\mathbf{s}$ allows us to drop the $l_{\mathbf{s}}$ term when ranking candidate summaries, since the score of all candidate summaries will receive an identical contribution from the $l_{\mathbf{s}}$ term. The experiments reported in the following section make this simplifying assumption.

[2]By incorporating a $p_{\mathbf{d}}$ model into the relevance model, equation (5.5) has implicitly resurrected the dependence on $\mathbf{d}$ which was dropped, for the sake of simplicity, in deriving (5.1).

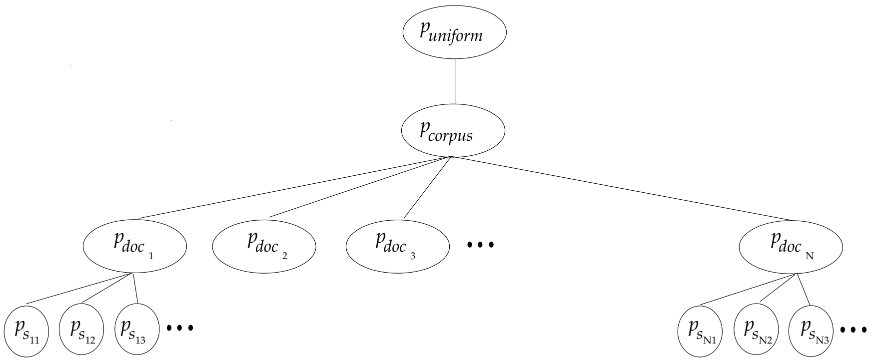


Figure 5.4: The relevance $p(\mathbf{q} \mid \mathbf{s}_{ij})$ of a query to the jth answer in document i is a convex combination of five distributions: (1) a uniform model $p_{\mathcal{U}}$. (2) a corpus-wide model $p_{\mathcal{C}}$; (3) a model $p_{\mathbf{d}_i}$ constructed from the document containing $\mathbf{s}_{ij}$; (4) a model $p_{\mathcal{N}_{ij}}$ constructed from $\mathbf{s}_{ij}$ and the neighboring sentences in $\mathbf{d}_i$; (3) a model $p_{\mathbf{s}_{ij}}$ constructed from $\mathbf{s}_{ij}$ alone. (The $p_{\mathcal{N}}$ distribution is omitted for clarity.)

5.3 Experiments

To gauge how well our proposed summarization technique performs, we applied it to two different real-world collections of answered questions:

Usenet FAQs: A collection of 201 frequently-asked question documents from the `comp.*` Usenet hierarchy. The documents contained 1800 questions/answer pairs in total.

Call-center data: A collection of questions submitted by customers to the companies Air Canada, Ben and Jerry, Iomagic, and Mylex, along with the answers supplied by company representatives. Among them, the four documents contain $10,395$ question/answer pairs. This is a superset of the dataset used in Chapter 3.8.

These datasets made an appearance in Section 3.8, in the context of answer-finding using statistical retrieval.

This section reports on an identical, parallel set of cross-validated experiments on both datasets. The first step was to use a randomly-selected subset of 70% of the question/answer pairs to calculate the language models $p_{\mathbf{s}}, p_{\mathcal{N}}, p_{\mathbf{d}}, p_{\mathcal{C}}$—a simple matter of counting word frequencies. The second step was to use this same set of data to estimate the model weights $\lambda = \{\lambda_{\mathbf{s}}, \lambda_{\mathcal{N}}, \lambda_{\mathbf{d}}, \lambda_{\mathcal{C}}, \lambda_{\mathcal{U}}\}$ using shrinkage, reserving the remaining 30% of the question/answer pairs to evaluate the performance of the system, in a manner described below.

Figure 5.5 shows the progress of the EM algorithm in calculating maximum-likelihood values for the smoothing coefficients λ, for the first of the three runs on the Usenet data. The

Algorithm 10: *Shrinkage for $\vec{\lambda}$ estimation*

Input: `Distributions` $p_{\mathbf{s}}, p_{\mathbf{d}}, p_{\mathcal{C}}, p_{\mathcal{U}}$,

$\mathcal{H} = \{\mathbf{d}, \mathbf{q}, \mathbf{s}\}$ `(not used to estimate` $p_{\mathbf{s}}, p_{\mathbf{d}}, p_{\mathcal{C}}, p_{\mathcal{U}}$`)`

Output `Model weights` $\vec{\lambda} = \{\lambda_{\mathbf{s}}, \lambda_{\mathcal{N}}, \lambda_{\mathbf{d}}, \lambda_{\mathcal{C}}, \lambda_{\mathcal{U}}\}$

1. `Set` $\lambda_{\mathbf{s}} \leftarrow \lambda_{\mathcal{N}} \leftarrow \lambda_{\mathbf{d}} \leftarrow \lambda_{\mathcal{C}} \leftarrow \lambda_{\mathcal{U}} \leftarrow 1/5$
2. `Repeat until` $\vec{\lambda}$ `converges:`
3. `Set` $\text{count}_{\mathbf{s}} = \text{count}_{\mathcal{N}} = \text{count}_{\mathbf{d}} = \text{count}_{\mathcal{C}} = \text{count}_{\mathcal{U}} = 0$
4. `Do for all` $\{\mathbf{d}, \mathbf{q}, \mathbf{s}\} \in \mathcal{H}$
5. `(`*`E-step`*`)` $\text{count}_{\mathbf{s}} \leftarrow \text{count}_{\mathbf{s}} + \frac{\lambda_{\mathbf{s}} p_{\mathbf{s}}(\mathbf{q})}{p(\mathbf{q} \mid \mathbf{s})}$

   `(similarly for` $\text{count}_{\mathcal{N}}, \text{count}_{\mathbf{d}}$`,` $\text{count}_{\mathcal{C}}$`,` $\text{count}_{\mathcal{U}}$`)`
6. `(`*`M-step`*`) Set` $\lambda_{\mathbf{s}} \leftarrow \frac{\text{count}_{\mathbf{s}}}{\sum_i \text{count}_i}$ `(similarly for` $\lambda_{\mathcal{N}}, \lambda_{\mathbf{d}}, \lambda_{\mathcal{C}}, \lambda_{\mathcal{U}}$`)`

quick convergence and the final λ values were essentially identical for the other partitions of this dataset.

The call-center data's convergence behavior was similar, although the final λ values were quite different. Figure 5.6 shows the final model weights for the first of the three experiments on both datasets. For the Usenet FAQ data, the corpus language model is the best predictor of the query and thus receives the highest weight. This may seem counterintuitive; one might suspect that answer to the query ($\mathbf{s}$, that is) would be most similar to, and therefore the best predictor of, the query. But the corpus model, while certainly biased away from the distribution of words found in the query, contains (by construction) no zeros, whereas each summary model is typically very sparse.

In the call-center data, the corpus model weight is lower at the expense of a higher document model weight. This might arise from the fact that the documents in the Usenet data were all quite similar to one another in lexical content, in contrast to the call-center documents. As a result, in the call-center data the document containing $\mathbf{s}$ will appear much more relevant than the corpus as a whole.

Evaluating the performance of the trained QRS model involved the previously-unseen portion of the FAQ data as follows: For each test $(\mathbf{d}, \mathbf{q})$ pair, record how highly the system ranked the correct summary $\mathbf{s}^{\star}$—the answer to $\mathbf{q}$ in $\mathbf{d}$—relative to the other answers in $\mathbf{d}$.

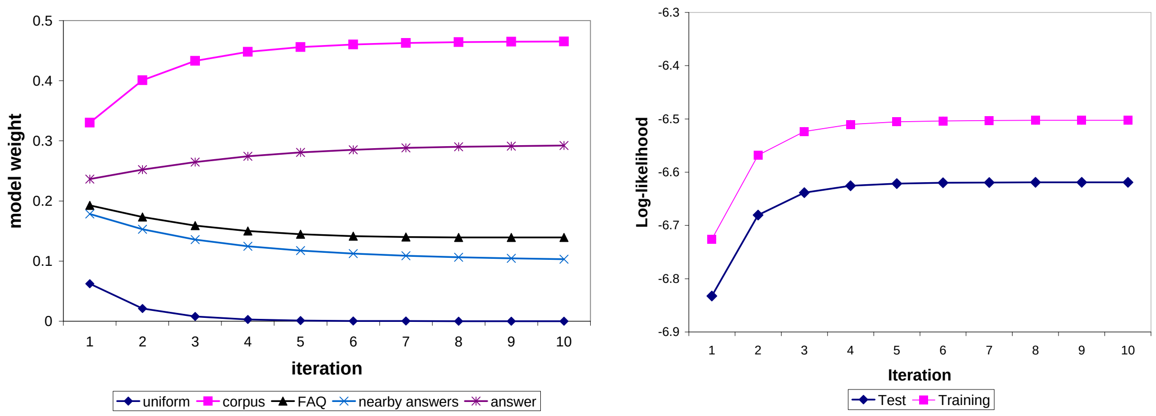


Figure 5.5: Estimating the weights of the five constituent models in (5.5) using the EM algorithm. The values here were computed using a single, randomly-selected 70% portion of the Usenet FAQ dataset. *Left*: The weights λ for the models are initialized to 1/5, but within a few iterations settle to their final values. *Right*: The progression of the likelihood of the training data during the execution of the EM algorithm; almost all of the improvement comes in the first five iterations.

Repeat this entire sequence three times for both the Usenet and the call-center data.

For these datasets, it turns out that using a uniform fidelity term in place of the $p(\mathbf{s} \mid \mathbf{d})$ model described above yields essentially the same result. This is not surprising: while the fidelity term is an important component of a real summarization system, the evaluation described here was conducted in an answer-locating framework, and in this context the fidelity term—enforcing that the summary be similar to the entire document from which it was drawn—is not so important.

Table 5.1 shows the inverse harmonic mean rank on the two collections. The third column of Table 5.1 shows the result of a QRS system using a uniform fidelity model, the fourth corresponds to a standard *tfidf*-based ranking method [67], and the last column reflects the performance of randomly guessing the correct summary from all answers in the document.

	$\lambda_{\mathbf{s}}$	$\lambda_{\mathcal{N}}$	$\lambda_{\mathbf{d}}$	$\lambda_{\mathcal{C}}$	$\lambda_{\mathcal{U}}$
Usenet FAQ	0.293	0.098	0.142	0.465	0
call-center	0.113	0.004	0.403	0.408	0.069

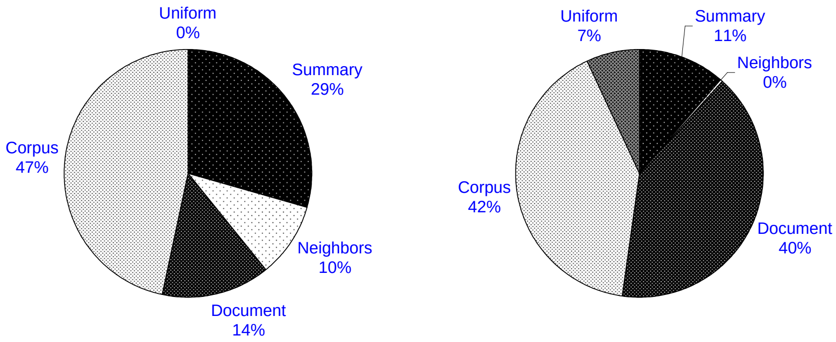


Figure 5.6: Maximum-likelihood weights for the various components of the relevance model $p(\mathbf{q} \mid \mathbf{s})$. *Left*: Weights assigned to the constituent models from the Usenet FAQ data. *Right*: Corresponding breakdown for the call-center data. These weights were calculated using shrinkage.

5.4 Extensions

5.4.1 Answer-finding

The reader may by now have realized that the QRS approach described here is applicable to the answer-finding task described in Section 3.8: automatically extracting from a potentially lengthy document (or set of documents) the answer to a user-specified question.

That section described how to use techniques from statistical translation to bridge the "lexical chasm" between questions and answers. This chapter, while focusing on the QRS problem, has incidentally mades two additional contributions to the answer-finding problem:

1. Dispensing with the simplifying assumption that the candidate answers are independent of one another by using a model which explicitly accounts for the correlation between text blocks—candidate answers—within a single document.

2. Proposing the use of FAQ documents as a proxy for query-summarized documents, which are difficult to come by.

Answer-finding and query-relevant summarization are, of course, not one and the same.

	trial	# questions	LM	tfidf	random
Usenet FAQ data	1	554	1.41	2.29	4.20
	2	549	1.38	2.42	4.25
	3	535	1.40	2.30	4.19
Call-center data	1	1020	4.8	38.7	1335
	2	1055	4.0	22.6	1335
	3	1037	4.2	26.0	1321

Table 5.1: Performance of query-relevant extractive summarization on the Usenet and call-center datasets. The numbers reported in the three rightmost columns are inverse harmonic mean ranks: lower is better.

For one, the criterion of containing an answer to a question is rather stricter than mere relevance. Put another way, only a small number of documents actually contain the answer to a given query, while every document can in principle be summarized with respect to that query. Second, it would seem that the $p(\mathbf{s} \mid \mathbf{d})$ term, which acts as a prior on summaries in (5.1), is less appropriate in a question-answering session: who cares if a candidate answer to a query doesn't bear much resemblance to the document containing it?

5.4.2 Generic extractive summarization

Although this chapter focuses on the task of query-relevant summarization, the core ideas—formulating a probabilistic model of the problem and learning the values of this model automatically from FAQ-like data—are equally applicable to generic summarization. In this case, one seeks the summary which best typifies the document. Applying Bayes' rule as in (5.1),

$$\begin{aligned} \mathbf{s}^\star &\equiv \arg\max_{\mathbf{s}} p(\mathbf{s} \mid \mathbf{d}) \\ &= \arg\max_{\mathbf{s}} \underbrace{p(\mathbf{d} \mid \mathbf{s})}_{\textit{generative}} \underbrace{p(\mathbf{s})}_{\textit{prior}} \end{aligned} \tag{5.6}$$

The first term on the right is a generative model of documents from summaries, and the second is a prior distribution over summaries. One can think of this factorization in terms of a dialogue. Alice, a newspaper editor, has an idea $\mathbf{s}$ for a story, which she relates to Bob. Bob researches and writes the story $\mathbf{d}$, which one can view as a "corruption" of Alice's original idea $\mathbf{s}$. The task of generic summarization is to recover $\mathbf{s}$, given only the generated document $\mathbf{d}$, a model $p(\mathbf{d} \mid \mathbf{s})$ of how the Alice generates summaries from documents, and a prior distribution $p(\mathbf{s})$ on ideas $\mathbf{s}$.

The central problem in information theory is reliable communication through an unreliable channel. In this setting, Alice's idea $\mathbf{s}$ is the original signal, and the process by which Bob turns this idea into a document $\mathbf{d}$ is the channel, which corrupts the original message. The summarizer's task is to "decode" the original, condensed message from the document. This is exactly the approach described in the last chapter, except that the summarization technique described there was non-extractive.

The factorization in (5.6) is superficially similar to (5.1), but there is an important difference: $p(\mathbf{d} \mid \mathbf{s})$ is a *generative*, from a summary to a larger document, whereas $p(\mathbf{q} \mid \mathbf{s})$ is *compressive*, from a summary to a smaller query.

5.5 Chapter summary

The task of summarization is difficult to define and even more difficult to automate. Historically, a rewarding line of attack for automating language-related problems has been to take a machine learning perspective: let a computer learn how to perform the task by "watching" a human perform it many times. This is the strategy adopted in this and the previous chapter.

In developing the QRS framework, this chapter has more or less adhered to the four-step strategy described in Chapter 1. Section 5.1 described how one can use FAQs to solve the problem of *data collection*. Section 5.2 introduced a family of statistical models for query-relevant summarization, thus covering the second step of *model selection*. Section 5.2 also covered the issue of *parameter estimation* in describing an EM-based technique for calculating the maximum-likelihood member of this family. Unlike in Chapter 4, search wasn't a difficult issue in this chapter—all that is required is to compute $p(\mathbf{s} \mid \mathbf{d}, \mathbf{q})$ according to (5.1) for each candidate summary $\mathbf{s}$ of a document $\mathbf{d}$.

There has been some work on learning a probabilistic model of summarization from text; some of the earliest work on this was due to Kupiec *et al.* [49], who used a collection of manually-summarized text to learn the weights for a set of features used in a generic summarization system. Hovy and Lin [40] present another system that learned how the position of a sentence affects its suitability for inclusion in a summary of the document. More recently, there has been work on building more complex, structured models—probabilistic syntax trees—to compress single sentences [47]. Mani and Bloedorn [55] have recently proposed a method for automatically constructing decision trees to predict whether a sentence should or should not be included in a document's summary. These previous approaches focus mainly on the generic summarization task, not query relevant summarization.

The language modelling approach described here does suffer from a common flaw within text processing systems: the problem of word relatedness. A candidate answer containing the term `Constantinople` is likely to be relevant to a question about Istanbul, but recognizing this correspondence requires a step beyond word frequency histograms. A natural extension of this work would be to integrate a word-replacement model as described in Section 3.8.

This chapter has proposed the use of two novel datasets for summarization: the frequently-asked questions (FAQs) from Usenet archives and question/answer pairs from the call centers of retail companies. Clearly this data isn't a perfect fit for the task of building a QRS system: after all, answers are not summaries. However, the FAQs appear to represent a reasonable source of query-related document condensations. Furthermore, using FAQs allows us to assess the effectiveness of applying standard statistical learning machinery—maximum-likelihood estimation, the EM algorithm, and so on—to the QRS problem. More importantly, it allows for a rigorous, non-heuristic evaluation of the system's performance. Although this work is meant as an opening salvo in the battle to conquer summarization with quantitative, statistical weapons, future work will likely enlist linguistic, semantic, and other non-statistical tools which have shown promise in condensing text.

Chapter 6

Conclusion

6.1 The four step process

Assessing relevance of a document to a query, producing a gist of a document, extracting a summary of a document relative to a query, and finding the answer to a question within a document: on the face of it, these appear to be a widely disparate group of problems in information management. The central purpose of this work, however, was to introduce and experimentally validate an approach, based on statistical machine learning, which applies to all of these problems.

The approach is the four-step process to statistical machine learning described in Chapter 1. With the full body of the thesis now behind us, it is worthwhile to recapitulate those steps:

- **Data collection**: One significant hurdle in using machine learning techniques to learn parametric models is finding a suitable dataset from which to estimate model parameters. It has been this author's experience that the data collection effort involves some amount of both imagination (to realize how a dataset can fulfill a particular need) and diplomacy (to obtain permission from the owner of the dataset to use it for a purpose it almost certainly wasn't originally intended for.)

  Chapters 3, 4 and 5 proposed novel datasets for learning to rank documents, summarize documents, and locate answers within documents. These datasets are, respectively, web portal "clickthrough" data, human-summarized web pages, and lists of frequently-asked question/answer pairs.

- **Model selection**: A common thread throughout this work is the idea of using parametric models adapted from those used in statistical translation to capture the word-relatedness effects in natural language. These models are essentially two-dimensional matrices of word-word "substitution" probabilities. Chapter 3 showed how this model can be thought of as an extension of two recently popular techniques in IR: language modeling and Hidden Markov Models (HMMs).

- **Parameter estimation**: From a large dataset of examples (of gisted documents, for instance), one can use the EM algorithm to compute the maximum-likelihood set of parameter estimates for that model.

- **Search**: In the case of answer-finding, "search" is a simple brute-force procedure: evaluate all candidate answers one by one, and take the best candidate. In the case of document ranking, the number of documents in question and the efficiency required in an interactive application preclude brute-force evaluation, and so this thesis has introduced a method for efficiently locating the most relevant document to a query while visiting only a small fraction of all candidate documents. The technique is somewhat reminiscent of the traditional IR expedient of using an inverse index. In the case of document gisting, the search space is exponential in the size of the generated summary, and so a bit more sophistication is required. Chapter 4 explains how one can use search techniques from artificial intelligence to find a high-scoring candidate summary quickly.

The promising empirical results reported herein do not indicate that "classic" IR techniques, like refined term-weighting formulae, query expansion, (pseudo)-relevance feedback, and stopword lists, are unnecessary. The opposite may in fact be true. For example, WEAVER relies on stemming (certainly a classic IR technique) to keep the matrix of synonymy probabilities of manageable size and ensure robust parameter estimates in spite of finitely-sized datasets. More generally, the accumulated wisdom of decades of research in document ranking is exactly what distinguishes mature document ranking systems in TREC evaluations year after year. One would not expect a system constructed entirely from statistical machine learning techniques to outperform these systems. An open avenue for future applied work in IR is to discover ways of integrating automatically-learned statistical models with well-established ad hoc techniques.

6.2 The context for this work

Pieces of the puzzle assembled in this document have been identified before. As mentioned above, teams from BBN and the University of Massachusetts have examined approaches to document ranking using language models and Hidden Markov Models [61, 67]. A group at Justsystem Research and Lycos Inc. [87] have examined automatic summarization using statistical translation.

In the case of document ranking, this thesis extends the University of Massachusetts and the BBN groups to intrinsically handle word-relatedness effects, which play a central role in information management. Chapter 3 includes a set of validating experiments on a heterogeneous collection of datasets including email, web pages, and newswire articles, establishing the broad applicability of document ranking systems built using statistical machine learning. Chapter 3 and subsequent chapters broaden the scope of this discovery to other problems in information processing, namely answer-finding and query-relevant summarization.

In the case of non-extractive summarization, Chapter 4 goes beyond previous work in explicitly factoring the problem into content selection and language modeling subtasks, and proposing a technique for estimating these models independently and then integrating them into a summarization algorithm which relies on stack search to identify an optimal summary. This work also represents the first attempt to apply non-extractive summarization to web pages, a natural domain because of the often disjointed nature of text in such documents.

6.3 Future directions

Over the course of this document appeared a number of avenues for further research. To recap, here are three particularly promising directions which apply not just to a single problem, but to several or all of the information processing problems discussed herein.

> **Polysemy**: WEAVER and OCELOT both attack the problem of word relatedness (or, loosely, "synonymy") through the use of statistical models parametrized by the probability that word x could appear in the place of word y. Knowing that a document containing the word `automobile` is relevant to a query containing the word `car` is a good start. But neither prototype directly addresses the equally important problem of *polysemy*—where a single word can have multiple meanings.
>
> For instance, the word `suit` has more than one sense, and a document containing this word is almost certainly relying on one of these senses. By itself, the word gives no hint as to which sense is most appropriate, but the surrounding words almost always elucidate the proper meaning. The task of *word sense disambiguation* is to analyze the context local to a word to decide which meaning is appropriate. There is a substantial body of work on automatic word-sense disambiguation algorithms, some of which employs statistical learning techniques [10], and it stands to reason

that such technology could improve the performance of WEAVER and OCELOT and the QRS prototype described earlier.

For instance, a "polysemy-aware" version of WEAVER could replace occurrences of the word `suit` in the legal sense with the new token $\texttt{suit}_1$, while replacing the word `suit` in the clothing sense with $\texttt{suit}_2$. The query `business suit` would then become `business` $\texttt{suit}_2$, and documents using `suit` in the clothing sense would receive a high ranking for this query, while those using the word in the legal sense would not. A similar line of reasoning suggests polysemy-awareness could help in summarization.

Discarding the independence assumption: Using local context to disambiguate the meaning of a word requires lifting the word independence assumption—the assumption that the order in which words appears in a document can be ignored. Of course, the idea that the order of words in a document is of no import is quite ludicrous. The two phrases `dog bites man` and `man bites dog` contain the same words, but have entirely different meanings.

By taking account of where words occur in a document, a text processing system can assign a higher priority to words appearing earlier in a document in the same way that people do. A document which explains in the first paragraph how to make an omelet, for instance, can be more valuable to a user than a document which waits until the ninth paragraph to do so.

Multilingual processing: Both the WEAVER and OCELOT systems are naturally applicable to a multilingual setting, where documents are in one language and queries (for WEAVER) or summaries (for OCELOT) are in another. This feature isn't pure serendipity; it exists because the architecture of both systems was inspired by earlier work in statistical translation. Finding high-quality multilingual text corpora and tailoring WEAVER and OCELOT for multilingual setting is a natural next step in the development of these systems.

* * *

There are compelling reasons to believe that the coming years will continue to witness an increase in the quality and prevalence of automatically-learned text processing systems.

For one, as the Internet continues to grow, so too will the data resources available to learn intelligent information processing behavior. For example, as mentioned in Chapter 4, recent work has described a technique for automatically discovering pairs of web pages written in two different languages—Chinese and English, say [73]. Such data could be used in learning a statistical model of translation. So as the number of web pages written in both Chinese and English increases, so too increases the raw material for building a Chinese-English translation system.

Second, so long as Moore's Law continues to hold true, the latest breed of computers will be able to manipulate increasingly sophisticated statistical models—larger vocabularies, more parameters, and more aggressive use of conditioning information.

Notes

Portions of Chapter 3 appeared in

A. Berger and J. Lafferty. The WEAVER system for document retrieval. *Proceedings of the Text Retrieval Conference (TREC-8)*, 1999.

A. Berger and J. Lafferty. Information retrieval as statistical translation. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1999.

A. Berger, R. Caruana, D. Cohn, D. Freitag and V. Mittal. Bridging the lexical chasm: Statistical approaches to answer-finding. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2000.

Portions of Chapter 4 appeared in

A. Berger and V. Mittal. OCELOT : A system for summarizing web pages. *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2000.

Portions of Chapter 5 appeared in

A. Berger and V. Mittal. Query-relevant summarization using FAQs. *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2000.

Bibliography

[1] M. Abramowitz and C. Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables.* Dover, 1972.

[2] R. Ash. *Information Theory.* Dover Publications, 1965.

[3] L. Bahl, F. Jelinek, and R. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5, 1983.

[4] L. Bahl and R. Mercer. Part of speech assignment by a statistical decision algorithm. *IEEE International Symposium on Information Theory (ISIT)*, 1976.

[5] L. Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. *Inequalities*, 3, 1972.

[6] A. Berger, P. Brown, S. Della Pietra, V. Della Pietra, J. Gillett, J. Lafferty, H. Printz, and L. Ures. The CANDIDE system for machine translation. In *Proceedings of the ARPA Human Language Technology Workshop*, 1994.

[7] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1999.

[8] A. Berger and J. Lafferty. The WEAVER system for document retrieval. In *Proceedings of the Text Retrieval Conference (TREC)*, 1999.

[9] A. Berger and R. Miller. Just-in-time language modelling. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1998.

[10] A. Berger, S. Della Pietra, and V. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), 1996.

[11] E. Black, F. Jelinek, J. Lafferty, and D. Magerman. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, 1992.

[12] P. Brown. *The acoustic modelling problem in automatic speech recognition.* PhD thesis, Carnegie Mellon University, 1987.

[13] P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, and P. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2), 1990.

[14] P. Brown, S. Della Pietra, V. Della Pietra, and R. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2), 1993.

[15] P. Brown, S. Della Pietra, V. Della Pietra, M. Goldsmith, J. Hajic, R. Mercer, and S. Mohanty. But dictionaries are data too. In *Proceedings of the ARPA Human Language Technology Workshop*, 1993.

[16] R. Burke, K. Hammond, V. Kulyukin, S. Lytinen, and N. Tomuro. Question answering from frequently-asked question files: Experiences with the FAQ Finder system. Technical Report TR-97-05, Department of Computer Science, University of Chicago, 1997.

[17] G. Cassella and R. Berger. *Statistical inference.* Brooks, Cole, 1990.

[18] Y. Chali, S. Matwin, and S. Szpakowicz. Query-biased text summarization as a question-answering technique. In *Proceedings of the AAAI Fall Symposium on Question Answering Systems*, 1999.

[19] C. Chelba. A structured language model. In *Proceedings of the ACL-EACL Joint Conference*, 1997.

[20] C. Chelba and F. Jelinek. Exploiting syntactic structure for language modeling. In *Proceedings of the Joint COLING-ACL Conference*, 1998.

[21] P. Clarkson and R. Rosenfeld. Statistical language modeling using the CMU-Cambridge toolkit. In *Eurospeech*, 1997.

[22] T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley and Sons, Inc., 1991.

[23] G. DeJong. An overview of the Frump system. In W. Lehnert and M. Ringle, editors, *Strategies for Natural Language Processing.* Lawrence Erlbaum Associates, 1982.

[24] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39B, 1977.

[25] S. Dumais, T. Letsche, M. Landauer, and M. Littman. Cross-language text and speech retrieval. In *AAAI Spring Symposium Series*, 1997.

[26] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[27] H. Edmundson. Problems in automatic extracting. *Communications of the ACM*, 7, 1964.

[28] E. Efthimiadis and P. Biron. UCLA-Okapi at TREC-2: Query expansion experiments. In *Proceedings of the Text Retrieval Conference (TREC)*, 1994.

[29] G. Forney. The Viterbi Algorithm. *Proceedings of the IEEE*, 1973.

[30] W. Francis and H. Kucera. A standard corpus of present-day edited American English, for use with digital computers. Technical report, Brown University Linguistics Department, 1961.

[31] M. Franz and J. McCarley. Machine translation and monolingual information retrieval. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1999.

[32] Gartner group report, 1998.

[33] J. Gemmell. ECRSM – Erasure Correcting Scalable Reliable Multicast. Technical report, Microsoft (TR 97-20), 1997.

[34] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell. Summarizing text documents: Sentence selection and evaluation metrics. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1999.

[35] I. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40, 1953.

[36] R. Gopinath. Personal communication, 1999.

[37] B. Greene and G. Rubin. Automatic grammatical tagging of English. Technical report, Department of Linguistics, Brown University, 1971.

[38] T. Hand. A proposal for task-based evaluation of text summarization systems. In *Proceedings of the ACL/EACL Workshop on Intelligent Scalable Text Summarization*, 1997.

[39] H. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14, 1953.

[40] E. Hovy and C. Lin. Automated text summarization in SUMMARIST. In *Proceedings of the ACL/EACL Workshop on Intelligent Scalable Text Summarization*, 1997.

[41] J. Ferguson, editor. *Symposium on the application of Hidden Markov Models to text and speech.* Institute for Defense Analysis, 1980.

[42] F. Jelinek. *Statistical methods for speech recognition.* MIT Press, 1997.

[43] F. Jelinek and R. Mercer. Interpolated estimation of Markov source parameters from sparse data. In *Workshop on Pattern Recognition in Practice*, 1980.

[44] H. Jing, R. Barzilay, K. McKeown, and M. Elhadad. Summarization evaluation methods experiments and analysis. In *Proceedings of the AAAI Intelligent Text Summarization Workshop*, 1998.

[45] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1987.

[46] M. Kearns and U. Vazirani. *An introduction to computational learning theory.* MIT Press, 1994.

[47] K. Knight and D. Marcu. Statistics-based summarization—Step one: Sentence compression. In *Proceedings of the International Conference on Artificial Intelligence (AAAI)*, 2000.

[48] S. Kullback. *Information Theory and Statistics.* Dover Publications, 1997.

[49] J. Kupiec, J. Pedersen, and F. Chen. A trainable document summarizer. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, July 1995.

[50] W. Lehnert. *The process of question answering: A computer simulation of cognition.* Lawrence Erlbaum Associates, 1978.

[51] D. Lewis and K. Knowles. Threading electronic mail: A preliminary study. *Information Processing and Management*, 33(2), 1997.

[52] P. Luhn. Automatic creation of literature abstracts. *IBM Journal*, 1958.

[53] D. Magerman. *Natural language parsing as statistical pattern recognition.* PhD thesis, Stanford University, 1994.

[54] J. Makhoul, R. Schwartz, C. LaPre, and I. Bazzi. A script-independent methodology for optical character recognition. *Pattern Recognition*, 31(9), 1998.

[55] I. Mani and E. Bloedorn. Machine learning of generic and user-focused summarization. In *Proceedings of the International Conference on Artificial Intelligence (AAAI)*, 1998.

[56] D. Marcu. From discourse structures to text summaries. In *Proceedings of the ACL/EACL Workshop on Intelligent Scalable Text Summarization*, 1997.

[57] M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank Project. *Computational Linguistics*, 19, 1993.

[58] M. Maron and J. Kuhns. On relevance, probabilistic indexing, and information retrieval. *Journal of the Association for Computing Machinery (JACM)*, 7, 1960.

[59] B. Mathis, J. Rush, and C. Young. Improvement of automatic abstracts by the use of structural analysis. *Journal of the American Society for Information Science*, 24, 1973.

[60] B. Merialdo. Tagging text with a probabilistic model. In *Proceedings of the IBM Natural Language ITL*, 1990.

[61] D. Miller, T. Leek, and R. Schwartz. A Hidden Markov Model information retrieval system. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1999.

[62] T. Mitchell. *Machine learning.* McGraw Hill, 1997.

[63] K. Nathan, H. Beigi, J. Subrahmonia, G. Clary, and H. Maruyama. Real-time on-line unconstrained handwriting recognition using statistical methods. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1995.

[64] J. Nie, M. Simard, P. Isabelle, and R. Durand. Cross-language information retrieval based on parallel texts and automatic mining of parallel texts from the web. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1999.

[65] D. Oard. Personal communication, 2000.

[66] The Open Directory project: `http://dmoz.org`, 1999.

[67] J. Ponte. *A language modelling approach to information retrieval.* PhD thesis, University of Massachusetts at Amherst, 1998.

[68] J. Ponte and W. Croft. A language modeling approach to information retrieval. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1998.

[69] A. Poritz. Hidden Markov Models: A guided tour. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1988.

[70] M. Porter. An algorithm for suffix stripping. *Program*, 14(3), 1980.

[71] D. Radev. Text summarization tutorial. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2000.

[72] A. Ratnaparkhi. A maximum entropy part of speech tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference (EMNLP'96)*, 1996.

[73] P. Resnick. Mining the Web for bilingual text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1999.

[74] S. Robertson and K. Sparck Jones. A probabilistic model for retrieval: development and status. Technical Report TR446, Cambridge University, 1998.

[75] J. Rocchio. Relevance feedback in information retrieval. In *The* SMART *retrieval system: Experiments in automatic document processing*, 1971.

[76] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24, 1988.

[77] S. Sato and M. Sato. Rewriting saves extracted summaries. In *Proceedings of the AAAI Intelligent Text Summarization Workshop*, 1998.

[78] R. Saunders. The thallium diagnostic workstation: Learning to diagnose heart injury from examples. In *Proceedings of the Second Innovative Applications of Artificial Intelligence Conference (IAAI'91)*, 1991.

[79] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948.

[80] C. Shannon. Prediction and entropy of written English. *Bell System Technical Journal*, 30, 1948.

[81] C. Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the Third Berkeley symposium on mathematical statistics and probability*, 1955.

[82] R. Szeliski. *Bayesian modelling of uncertainy in low-level vision.* Kluwer Academic Publishers, 1989.

[83] The Internet Archive. `http://www.archive.org`, 2000.

[84] E. Voorhees and D. Harman. Overview of the Eighth Text Retrieval Conference. In *Proceedings of the Text Retrieval Conference (TREC)*, 1999.

[85] F. Wancho. RFC 1153: Digest message format, 1990.

[86] W. Welch. (unpublished notes).

[87] M. Witbrock and V. Mittal. Headline generation: A framework for generating highly-condensed non-extractive summaries. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1999.

[88] C. Wu. On the convergence properties of the EM algorithm. *Annals of Statistics*, 11, 1983.

[89] J. Xu and B. Croft. Query expansion using local and global document analysis. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 1996.